Progress DataDirect for ODBC for Apache Cassandra

User's Guide

Release 8.0.0

Copyright

Table of Contents

1

Welcome to the Progress DataDirect for ODBC for Apache Cassandra Driver

The Progress® DataDirect® for ODBC™ for Apache Cassandra™ driver supports read-write access to Apache Cassandra versions. To support SQL access to Cassandra, the driver creates a relational map of native Cassandra data and translates SQL statements to CQL. Cassandra complex data types Map, List, Set, Tuple, and user-defined types are supported alongside primitive CQL types. The driver optimizes performance when executing joins by leveraging data relationships among Cassandra objects to minimize the amount of data that needs to be fetched over the network. Relationships among objects can be reported with the following metadata methods: SQLColumns, SQLForeignKeys, SQLGetTypeInfo, SQLPrimaryKeys, SQLSpecialColumns, SQLStatistics, and SQLTables. Furthermore, when performing joins, the driver leverages data relationships among Cassandra objects, minimizing the amount of data that needs to be fetched over the network.

The documentation for the driver also includes the *Progress DataDirect for ODBC Drivers Reference*. The reference provides general reference information for all DataDirect drivers for ODBC, including content on troubleshooting, supported SQL escapes, and DataDirect tools. For the complete documentation set, visit to the Progress DataDirect Connectors Documentation Hub: https://docs.progress.com/bundle/datadirect-connectors/page/DataDirect-Connectors-by-data-source.html.

For details, see the following topics:

What's new in this release?

Support and Certifications

Visit the following web pages for the latest support and certification information.

- Release Notes: https://www.progress.com/odbc/release-history/
- DataDirect Product Compatibility Guide: https://docs.progress.com/bundle/datadirect-product-compatibility/resource/datadirect-product-compatibility.pdf

Changes Since the 8.0.0 Release

- Enhancements
  - A Password Encryption Tool, called ddencpwd, is now included with the product package. It encrypts passwords for secure handling in connection strings and odbc.ini files. At connection, the driver decrypts these passwords and passes them to the data source as required. See Password Encryption Tool (UNIX/Linux only) on page 69 for details.
  - The driver has been enhanced to allow you to configure how collections are mapped in the relational view of your data. By configuring the new Schema Format (SchemaFormat) config option, you can determine whether the driver normalizes collections and collections labeled FROZEN. See SchemaFormat (config option) on page 94 for details.
  - The driver has been enhanced to support connection failover and client load balancing when connecting to a cluster. You can enable this new functionality by specifying a comma-separated list of member nodes using the new Cluster Nodes (ClusterNodes) connection option. See Using failover in a cluster on page 77 and Cluster Nodes on page 92 for details. Available in driver version 08.00.0216 (B0706, U0505, J000324).
  - The driver has been enhanced to include timestamp in the internal packet logs by default. If you want to disable the timestamp logging in packet logs, set `PacketLoggingOptions=1`. The internal packet logging is not enabled by default. To enable it, set `EnablePacketLogging=1`.
  - The driver has been enhanced to support the Duration data type, which maps to the SQL_VARCHAR ODBC type. See Data types on page 15 for details.
  - The driver has been enhanced to support all the data consistency levels for read and write operations that are supported by Apache Cassandra data stores. Data consistency levels are configured using the Read Consistency and Write Consistency connection options. For additional information, see Read Consistency on page 108 and Write Consistency on page 117.

Highlights of the 8.0.0 Release

- Certified against Apache Cassandra versions 2.0, 2.1, 2.2, and 3.7.
- Certified against DataStax Enterprise 4.7, 4.8, and 5.0.

- Supports SQL read-write access to Apache Cassandra data sources. See Supported SQL functionality on page 119 for details.
- Supports CQL Binary Protocol.
- The driver supports the core SQL-92 grammar.
- Support for all ODBC Core and Level 1 functions and some Level 1 and Level 2 features. See ODBC compliance on page 13 for details.
- Supports user ID and password authentication. See Authentication Method on page 92 for details.
- Supports Cassandra data types, including the complex types Tuple, user-defined types, Map, List and Set. See Data types on page 15 for details.
- Generates a relational view of Cassandra data. Tuple and user-defined types are flattened into a relational parent table, while collection types are mapped as relational child tables. See Complex type normalization on page 18 for details.
- Supports Native and Refresh escape clauses to embed CQL commands in SQL-92 statements. See Native and Refresh escape clauses on page 120 for details.
- Supports Cassandra's tunable consistency functionality with Read Consistency on page 108 and Write Consistency on page 117 connection options.
- Supports the handling of large result sets with Fetch Size on page 98, Native Fetch Size on page 106, and Result Memory Size on page 110 connection options.
- Supports applications that do not support unbounded Cassandra data types through the Ascii Size on page 91, Decimal Precision on page 96, Decimal Scale on page 97, Varchar Size on page 116, and Varint Precision on page 116 connection options.

Driver requirements

Data source and platform requirements

For the latest support information, visit the DataDirect Product Compatibility Guide: https://docs.progress.com/bundle/datadirect-product-compatibility/resource/datadirect-product-compatibility.pdf.

Java requirements

- The driver requires a Java Virtual Machine (JVM) that is Java SE 8 or higher. JVM support includes Oracle JDK, OpenJDK, and IBM SDK (Java) distributions.
- For 32-bit drivers, a 32-bit Java Virtual Machine (JVM) is required. For 64-bit drivers, a 64-bit Java Virtual Machine (JVM) is required.
- For Windows, you must set the PATH environment variable to the directory containing the `jvm.dll` for your JVM.
- For UNIX/Linux, you must set the library path environment variable of your operating system to the directory containing your JVM's `libjvm.so` file and that directory's parent directory.

Windows requirements for 32-bit drivers

- All required network software that is supplied by your database system vendors must be 32-bit compliant.
- You must have Microsoft Visual C/C++ runtime version 14.40.33810 or higher.
- You must have ODBC header files to compile your application. For example, Microsoft Visual Studio includes these files.

Windows requirements for 64-bit drivers

- All required network software that is supplied by your database system vendors must be 64-bit compliant.
- You must have Microsoft Visual C/C++ runtime version 14.40.33810 or higher.
- You must have ODBC header files to compile your application. For example, Microsoft Visual Studio includes these files.

Linux requirements for 32-bit drivers

- If your application was built with 32-bit system libraries, you must use 32-bit drivers. The database to which you are connecting can be either 32-bit or 64-bit enabled.
- An application compatible with components that were built using g++ GNU project C++ Compiler version 3.4.6 and the Linux native pthread threading model (Linuxthreads).

Linux requirements for 64-bit drivers

- An application compatible with components that were built using g++ GNU project C++ Compiler version 3.4 and the Linux native pthread threading model (Linuxthreads).

AIX requirements for 32-bit and 64-bit drivers

- IBM POWER processor
- An application compatible with components that were built using Visual Age C++ 6.0.0.0 and the AIX native threading model.

HP-UX requirements for 32-bit drivers

- The following processors are supported:
  - PA-RISC
  - Intel Itanium II (IPF)
- For PA-RISC: An application compatible with components that were built using HP aC++ 3.30 and the HP-UX 11 native (kernel) threading model (posix draft 10 threads).
- For IPF: An application compatible with components that were built using HP aC++ 5.36 and the HP-UX 11 native (kernel) threading model (posix draft 10 threads).

HP-UX requirements for 64-bit drivers

- Intel Itanium II (IPF) processor
- HP aC++ v. 5.36 and the HP-UX 11 native (kernel) threading model (posix draft 10 threads).

Oracle Solaris requirements for 32-bit drivers

- The following processors are supported:
  - Oracle SPARC
  - x86: Intel
  - x64: Intel and AMD
- For Oracle SPARC: An application compatible with components that were built using Oracle Workshop version 6 update 2 and the Solaris native (kernel) threading model.
- For x86/x64: An application compatible with components that were built using Oracle C++ 5.8 and the Solaris native (kernel) threading model.

Oracle Solaris requirements for 64-bit drivers

- The following processors are supported:
  - Oracle SPARC
  - x64: Intel and AMD
- For Oracle SPARC: An application compatible with components that were built using Oracle Workshop version 6 update 2 and the Solaris native (kernel) threading model.
- For x64: An application compatible with components that were built using Oracle C++ Compiler version 5.8 and the Solaris native (kernel) threading model.

ODBC compliance

The driver is compliant with the Open Database Connectivity (ODBC) 3.52 specification. The driver is ODBC core compliant and supports some Level 1 and Level 2 features.

The driver supports only the following Level 2 functions:

- SQLColumnPrivileges
- SQLDescribeParam
- SQLForeignKeys
- SQLPrimaryKeys
- SQLProcedures
- SQLTablePrivileges

Refer to "ODBC API and scalar functions" in the *Progress DataDirect for ODBC Drivers Reference* for additional information.

Version string information

The driver for Apache Cassandra has a version string of the format:

XX.YY.ZZZZ(B*AAAA*, U*BBBB*, J*DDDDDD*)

The Driver Manager on UNIX and Linux has a version string of the format:

XX.YY.ZZZZ(U*BBBB*)

The component for the Unicode conversion tables (ICU) has a version string of the format:

XX.YY.ZZZZ

where:

XX is the major version of the product.

YY is the minor version of the product.

ZZZZ is the build number of the driver or ICU component.

AAAA is the build number of the driver's bas component.

BBBB is the build number of the driver's utl component.

DDDDDD is the version of the Java components used by the driver.

For example:

```
08.00.0017 (B0254, U0180, J000109)
      |__|  |___|  |___|   |____|
     Driver  Bas    Utl     Java
```

On Windows, you can check the version string through the properties of the driver DLL. Right-click the driver DLL and select **Properties**. The Properties dialog box appears. On the Details tab, the **File Version** will be listed with the other file properties.

You can always check the version string of a driver on Windows by looking at the About tab of the driver's Setup dialog.

UNIX®

On UNIX and Linux, you can check the version string by using the test loading tool shipped with the product. This tool, ivtestlib for 32-bit drivers and ddtestlib for 64-bit drivers, is located in *install_directory*/bin.

The syntax for the tool is:

```
ivtestlib shared_object
```

or

```
ddtestlib shared_object
```

For example, for the 32-bit driver on Oracle Solaris:

```
ivtestlib ivcsndr28.so
```

returns:

```
08.00.0001 (B0002, U0001, J000003)
```

For example, for the Driver Manager on Solaris:

```
ivtestlib libodbc.so
```

returns:

```
08.00.0001 (U0001)
```

For example, for the 64-bit Driver Manager on Solaris:

```
ddtestlib libodbc.so
```

returns:

```
08.00.0001 (U0001)
```

For example, for the 32-bit ICU component on Solaris:

```
ivtestlib libivicu28.so
08.00.0001
```

Note: On AIX, Linux, and Solaris, the full path to the driver does not have to be specified for the test loading tool. The HP-UX version of the tool, however, requires the full path.

getFileVersionString function

Version string information can also be obtained programmatically through the function `getFileVersionString`. This function can be used when the application is not directly calling ODBC functions.

This function is defined as follows and is located in the driver's shared object:

```
const unsigned char* getFileVersionString();
```

This function is prototyped in the `qesqlext.h` file shipped with the product.

Data types

The following table lists the Apache Cassandra data types and their default mapping for ODBC.

Table 1: Default Mapping for Apache Cassandra Data Types

Apache Cassandra Data Type	ODBC Data Type
ASCII[1]	SQL_VARCHAR (12)
Bigint	SQL_BIGINT (-5)
Blob	SQL_LONGVARBINARY (-4)

[1] ASCII precision is set to 4000 by default, but you can use the AsciiSize connection option to configure Ascii precision. See "Ascii Size" for details.

Apache Cassandra Data Type	ODBC Data Type
Boolean	SQL_BIT (-7)
Counter[2]	SQL_BIGINT (-5)
Date	SQL_TYPE_DATE (91)
Decimal[3]	SQL_DECIMAL (3)
Double	SQL_DOUBLE (8)
Duration[4]	SQL_VARCHAR (12)
Float	SQL_REAL (7)
Inet	SQL_VARCHAR (12)
Int	SQL_INTEGER (4)
List	SQL_WLONGVARCHAR (-10)
Map	SQL_WLONGVARCHAR (-10)
Set	SQL_WLONGVARCHAR (-10)
Smallint	SQL_SMALLINT (5)
Time	SQL_TYPE_TIME (92)
Timestamp	SQL_TYPE_TIMESTAMP (93)
TimeUUID	SQL_CHAR (1)
TinyInt	SQL_TINYINT (-6)
Tuple	SQL_WLONGVARCHAR (-10)
Usertype	SQL_WLONGVARCHAR (-10)
UUID	SQL_CHAR (1)
Varchar[5]	SQL_WVARCHAR (9)
Varint[6]	SQL_DECIMAL (3)

[2] Update is supported for Counter columns when all the other columns in the row comprise that row's primary key. See "Update" for details.

[3] By default, Decimal precision is set to 38 and scale is set to 10; however, you can use the DecimalPrecision and DecimalScale connection options to configure Decimal precision and scale. See "Decimal Precision" and "Decimal Scale" for details.

[4] Currently, the Duration type is supported only in simple columns, and not in Collection types.

[5] Varchar precision is set to 4000 by default, but you can use the VarcharSize connection option to configure Varchar precision. See "Varchar Size" for details.

[6] Varint precision is set to 38 by default, but you can use the VarintPrecision connection option to configure Varint precision. See "Varint Precision" for details.

See also

Ascii Size on page 91
Update on page 134
Decimal Precision on page 96
Decimal Scale on page 97
Varchar Size on page 116
Varint Precision on page 116

Retrieving data type information

At times, you might need to get information about the data types that are supported by the data source, for example, precision and scale. You can use the ODBC function SQLGetTypeInfo to do this.

On Windows, you can use ODBC Test to call SQLGetTypeInfo against the ODBC data source to return the data type information.

Refer to "Diagnostic tools" in the *Progress DataDirect for ODBC Drivers Reference* for details about ODBC Test.

On all platforms, an application can call SQLGetTypeInfo. Here is an example of a C function that calls SQLGetTypeInfo and retrieves the information in the form of a SQL result set.

```
void ODBC_GetTypeInfo(SQLHANDLE hstmt, SQLSMALLINT dataType)
{
    RETCODE rc;

// There are 19 columns returned by SQLGetTypeInfo.
// This example displays the first 3.
// Check the ODBC 3.x specification for more information.
// Variables to hold the data from each column
    char             typeName[30];
    short            sqlDataType;
    unsigned int     columnSize;

    SQLLEN           strlenTypeName,
                     strlenSqlDataType,
                     strlenColumnSize;

    rc = SQLGetTypeInfo(hstmt, dataType);
    if (rc == SQL_SUCCESS) {

// Bind the columns returned by the SQLGetTypeInfo result set.
        rc = SQLBindCol(hstmt, 1, SQL_C_CHAR, &typeName,
              (SDWORD)sizeof(typeName), &strlenTypeName);
        rc = SQLBindCol(hstmt, 2, SQL_C_SHORT, &sqlDataType,
              (SDWORD)sizeof(sqlDataType), &strlenSqlDataType);
        rc = SQLBindCol(hstmt, 3, SQL_C_LONG, &columnSize,
              (SDWORD)sizeof(columnSize), &strlenColumnSize);

// Print column headings
         printf ("TypeName      DataType          ColumnSize\n");
         printf ("--------------------- ---------- ----------\n");

        do {

// Fetch the results from executing SQLGetTypeInfo
            rc = SQLFetch(hstmt);
            if (rc == SQL_ERROR) {
// Procedure to retrieve errors from the SQLGetTypeInfo function
               ODBC_GetDiagRec(SQL_HANDLE_STMT, hstmt);
               break;
              }

// Print the results
         if ((rc == SQL_SUCCESS) || (rc == SQL_SUCCESS_WITH_INFO)) {
printf ("%-30s %10i %10u\n", typeName, sqlDataType, columnSize);
             }

        } while (rc != SQL_NO_DATA);
    }
}
```

Complex type normalization

To support SQL access to Apache Cassandra, the driver maps the Cassandra data model to a relational schema. This process involves the normalization of complex types. You may need to be familiar with the normalization of complex types to formulate SQL queries correctly. The driver handles the normalization of complex types in the following manner:

- If collection types (Map, List, and Set) are discovered, the driver normalizes the Cassandra table into a set of parent-child tables. Primitive types are mapped to a parent table, while each collection type is mapped to a child table that has a foreign key relationship to the parent table.

- Non-nested Tuple and user-defined types (also referred to as *Usertype*) are flattened into a parent table alongside primitive types.
- Any nested complex types (Tuple, user-defined types, Map, List, and Set) are exposed as JSON-style strings in the parent table.

The normalization of complex types is described in greater detail in the following topics.

Note: This section describes the default mapping behavior of the driver. You can configure this behavior using the SchemaFormat configuration option. See SchemaFormat (config option) on page 94 for details.

Collection types

Cassandra collection types include the Map, List, and Set types. If collection types are discovered, the driver normalizes the native data into a set of parent-child tables. Primitive types are normalized in a parent table, while each collection type is normalized in a child table that has a foreign key relationship to the parent table. Take for example the following Cassandra table:

```
CREATE TABLE employee (
   empid int PRIMARY KEY,
   phone map<varchar, varint>,
   client list<varchar>,
   review set<date>);
```

The following `employee` table is a tabular representation of the native Cassandra table with data included. In this example, four distinct relational tables are created. A parent table is created based on the `empid` column, and a child table is created for each of the three collection types (Map, List, and Set).

Table 2: employee (native)

empid (primary key)	phone	client	review
int	map<varchar, varint>	list<varchar>	set<date>
103	home: 2855551122 mobile: 2855552347 office: 2855555566 spouse: 2855556782	Li Kumar Jones	2013-12-07 2015-01-22 2016-01-10
105	home: 2855555678 mobile: 2855553335 office: 2855555462	Yanev Bishop Bogdanov	2015-01-22 2016-01-12

The Parent Table

The parent table is comprised of the primitive integer type column `EMPID` and takes its name from the native table. A SQL statement would identify the column as `EMPLOYEE.EMPID`.

Table 3: EMPLOYEE (relational parent)

EMPID (primary key)
int
103
105

A SQL insert on the EMPLOYEE parent table would take the form:

```
INSERT INTO EMPLOYEE (EMPID) VALUES (107)
```

The Map Child Table

The Map collection is normalized into a three column child table called EMPLOYEE_PHONE. The name of the table is formulated by concatenating the name of the native table and the name of the Map column. A foreign key relationship to the parent table is maintained via the EMPLOYEE_EMPID column, and the Map's key value pairs are resolved into separate KEYCOL and VALUECOL columns. In a SQL statement, these columns would be identified as the EMPLOYEE_PHONE.EMPLOYEE_EMPID, EMPLOYEE_PHONE.KEYCOL, and EMPLOYEE_PHONE.VALUECOL, respectively.

Table 4: EMPLOYEE_PHONE (relational child of the map column)

EMPLOYEE_EMPID (foreign key)	KEYCOL	VALUECOL
int	varchar	varint
103	home	2855551122
103	mobile	2855552347
103	office	2855555566
103	spouse	2855556782
105	home	2855555678
105	mobile	2855553335
105	office	2855555462

A SQL insert on the EMPLOYEE_PHONE child table would take the form[7]:

```
INSERT INTO EMPLOYEE_PHONE (EMPLOYEE_EMPID, KEYCOL, VALUECOL)
   VALUES (107, 'mobile', 2855552391)
```

The List Child Table

The List collection is normalized into a three column child table called EMPLOYEE_CLIENT. The name of the table is formulated by concatenating the name of the native table and the name of the List column. A foreign key relationship to the parent table is maintained via the EMPLOYEE_EMPID column; the order of the elements in the List is maintained via the CURRENT_LIST_INDEX column; and the elements themselves are contained in the CLIENT column. SQL statements would identify these columns as EMPLOYEE_CLIENT.EMPLOYEE_EMPID, EMPLOYEE_CLIENT.CURRENT_LIST_INDEX, and EMPLOYEE_CLIENT.CLIENT, respectively.

Table 5: EMPLOYEE_CLIENT (relational child of the list column)

EMPLOYEE_EMPID (foreign key)	CURRENT_LIST_INDEX	CLIENT
int	int	varchar
103	0	Li
103	1	Kumar
103	2	Jones
105	0	Yanev
105	1	Bishop
105	2	Bogdanov

A SQL insert on the EMPLOYEE_CLIENT child table would take the form[7]:

```
INSERT INTO EMPLOYEE_CLIENT (EMPLOYEE_EMPID, CLIENT) VALUES (107, 'Nelson')
```

The Set Child Table

The Set collection is normalized into a two column child table called EMPLOYEE_REVIEW. The name of the table is formulated by concatenating the name of the native table and the name of the Set column. A foreign key relationship to the parent table is maintained via the EMPLOYEE_EMPID column, while the elements of the Set are given in natural order in the REVIEW column. In this child table, SQL statements would identify these columns as EMPLOYEE_REVIEW.EMPLOYEE_EMPID and EMPLOYEE_REVIEW.REVIEW

[7] The driver supports an insert on a child table prior to an insert on a parent table, circumventing referential integrity constraints associated with traditional RDBMS. To maintain integrity between parent and child tables, it is recommended that an insert first be performed on the parent table for each foreign key value added to the child. If such an insert is not first performed, the driver automatically inserts a row into the parent table that contains only the primary key values and NULL values for all non-primary key columns.

Table 6: EMPLOYEE_REVIEW (relational child of the set column)

EMPLOYEE_EMPID (foreign key)	REVIEW
int	date
103	2013-12-07
103	2015-01-22
103	2016-01-10
105	2015-01-22
105	2016-01-12

A SQL insert on the EMPLOYEE_CLIENT child table would take the form[7]:

```
INSERT INTO EMPLOYEE_REVIEW (EMPLOYEE_EMPID, REVIEW) VALUES (107, '2015-01-20')
```

Update Support

Update is supported for primitive types, non-nested Tuple types, and non-nested user-defined types. Update is also supported for value columns (VALUECOL) in non-nested Map types. The driver does not support updates on List types, Set types, or key columns (KEYCOL) in Map types because the values in each are part of the primary key of their respective child tables and primary key columns cannot be updated. If an Update is attempted when not allowed, the driver issues the following error message:

```
[DataDirect][Cassandra ODBC Driver][Cassandra]syntax error or access rule violation:
UPDATE not permitted for column: column_name
```

Tuple and user-defined types

The driver supports Tuple and user-defined complex types which were introduced with Apache Cassandra 2.1. As long as there are no complex types nested in either the Tuple or user-defined types, the driver normalizes Tuple and user-defined types by flattening them into a relational version of the native Cassandra table. Take for example the following Cassandra table:

```
CREATE TABLE agents1 (
   agentid int PRIMARY KEY,
   email varchar,
   contact tuple<varchar,varchar,varchar>);
```

The following AGENTS1 table is a tabular representation of the native Cassandra table with data included.

Table 7: AGENTS1 (native)

AGENTID (primary key)	EMAIL	CONTACT
int	varchar	tuple<varchar, varchar, varchar>
272	barronr@email.com	tv newspaper blog
564	rothm@email.com	radio tv magazine

For the relational version of `agents1`, all fields are retained as separate columns, and columns with primitive types (`AGENTID` and `EMAIL`) correspond directly to columns in the native table. In turn, tuple fields are flattened into columns using a `<tuplename>_<ordinal>` naming pattern. The driver normalizes `agents1` in the following manner.

Table 8: AGENTS1 (relational)

AGENTID (primary key)	EMAIL	CONTACT_1	CONTACT_2	CONTACT_3
int	varchar	varchar	varchar	varchar
272	barronr@email.com	tv	newspaper	blog
564	rothm@email.com	radio	tv	magazine

A SQL command would take the following form:

```
INSERT INTO AGENTS1 (AGENTID,EMAIL,CONTACT_1,CONTACT_2,CONTACT_3)
   VALUES (839,'gonzalesn@email.com','radio','tv','magazine')
```

The driver also flattens user-defined types when normalizing native Cassandra tables. In the following example, the native Cassandra `agents2` table incorporates the user-defined *address* type.

```
CREATE TYPE address (
   street varchar,
   city varchar,
   state varchar,
   zip int);
CREATE TABLE agents2 (
   agentid int PRIMARY KEY,
   email varchar,
   location frozen<address>);
```

The following `AGENTS2` table is a tabular representation of the native Cassandra table with data included.

Table 9: AGENTS2 (native)

AGENTID (primary key)	EMAIL	LOCATION
int	varchar	address<street: varchar, city: varchar, state: varchar, zip: int>
095	barronr@email.com	street: 1551 Main Street city: Pittsburgh state: PA zip: 15237
237	rothm@email.com	street: 422 First Street city: Richmond state: VA zip: 23235

As with the previous example, all fields are retained as separate columns in the relational version of the table, and columns with primitive types (`agentid` and `email`) correspond directly to columns in the native table. Here a `<columnname>_<fieldname>` naming pattern is used to flatten the fields of the user-defined address type into columns. The driver normalizes `agents2` in the following manner.

Table 10: AGENTS2 (native)

AGENTID (primary key)	EMAIL	LOCATION_STREET	LOCATION_CITY	LOCATION_STATE	LOCATION_ZIP
int	varchar	varchar	varchar	varchar	int
095	barronr@email.com	1551 Main Street	Pittsburgh	PA	15237
237	rothm@email.com	422 First Street	Richmond	VA	23235

A SQL command would take the following form:

```
INSERT INTO AGENTS2
(AGENTID,EMAIL,LOCATION_STREET,LOCATION_CITY,LOCATION_STATE,LOCATION_ZIP)
   VALUES (839,'gonzalesn@email.com','9 Fifth Street', 'Morrisville', 'NC', 27566)
```

Nested complex types

The nesting of complex types within Tuple and user-defined types is permitted in CQL. The driver does not normalize such nested types, but rather the data is passed as a JSON-style string. For example, consider the table `contacts` which contains the columns `id` and `contact`. While `id` is a primitive int column, `contact` is a user-defined *info* column which contains `name`, `email`, and `location` fields. The `location` field itself is a nested user-defined *address* column which contains `street`, `city`, `state`, and `zip` fields. In CQL, the structure of this table would take the following form:

```
CREATE TYPE address (
   street varchar,
   city varchar,
   state varchar,
   zip int);
CREATE TYPE info (
   name varchar,
   email varchar,
   location frozen<address>);
CREATE TABLE contacts (
   id int PRIMARY KEY,
   contact frozen<info>);
```

The following tabular representation of the `contacts` table shows how the driver returns data when complex types are nested in other complex types. Because the complex user-defined type `address` is embedded in the complex user-defined type `info`, the entire `CONTACT` column is returned by the driver as a JSON string.

Note: You can retrieve this string data by calling the DatabaseMetaData.getColumns() method.

Table 11: CONTACTS (relational)

ID (primary key)	CONTACT
int	info<name: varchar, email: varchar, location: address<street: varchar, city: varchar, state: varchar, zip: int>>
034	{name: 'Jude', email: 'jnichols@email.com', location: {street: '101 Main Street', city: 'Albany', state:'NY', zip: 12210}}
056	{name: 'Karen', email: 'kbrown@email.com', location: {street: '150 First Street', city: 'Portland', state: 'OR', zip: 97214}}

When executing SQL commands involving nested complex types, the data must be passed as a JSON string. Furthermore, the syntax you use to connote the JSON string depends on whether you are passing the string directly in a SQL command or binding the JSON string as a parameter to a variable in the application.

Note: Hints for parsing JSON-style strings are provided in the `REMARK` column of the getColumns() result.

Connoting the JSON-Style String in a SQL Statement

When passing the string directly in a SQL command, you must use the correct SQL syntax and escapes to maintain the structure of the data. To begin, the entire JSON string must be passed in single quotation marks ('). Furthermore, if the JSON string contains nested strings, two single quotation marks are used to indicate string values. The first quotation mark is an escape connoting the second embedded quotation mark. The following command inserts a new row into the `CONTACTS` table.

Note: In accordance with typical programming language syntax, the Insert statement is placed in double quotation marks. However, in the JSON string, two single quotation marks are used to indicate string values. The first quotation mark is an escape connoting the second embedded quotation mark.

```
SQLExecDirect(
   pstmt,
   "INSERT INTO CONTACTS (ID, CONTACT) VALUES (075, '{name: ''Albert'',
     email: ''aocampo@email.com'', location: {street: ''12 North Street'',
     city: ''Durham'', state:''NC'', zip: 27704}}')",
   SQL_NTS);
```

After the insert has been executed, the Select command `SELECT CONTACT FROM CONTACTS WHERE ID = 75` returns:

```
{name: 'Albert',
email: 'aocampo@email.com',
location: {street: '12 North Street',
           city: 'Durham',
           state:'NC',
           zip: 27704
           }
}
```

Connoting the JSON-Style String as a Parameter Variable

When binding the JSON string as a parameter to a variable in the application, you must follow your programming language syntax by placing the JSON string in double quotation marks. Escapes are not used to connote embedded single quotation marks. For example:

```
STRING string_variable =
   "{name: 'Albert', email: 'aocampo@email.com',
      location: {street: '12 North Street',
      city: 'Durham', state:'NC', zip: 27704}}"
```

SQL support

The Apache Cassandra driver provides support for standard SQL (primarily SQL 92), and a set of SQL extensions.

See also

Supported SQL functionality on page 119

Additional information

In addition to the content provided in this guide, the documentation set also contains detailed conceptual and reference information that applies to all the drivers. For more information in these topics, refer the *Progress DataDirect for ODBC Drivers Reference* or use the links below to view some common topics:

- "Code page values" lists supported code page values, along with a description, for the Progress DataDirect for ODBC drivers.
- "ODBC API and scalar functions" lists the ODBC API functions supported by Progress DataDirect for ODBC drivers. In addition, it documents the scalar functions that you use in SQL statements.
- "Internationalization, localization, and Unicode" provides an overview of how internationalization, localization, and Unicode relate to each other. It also includes a background on Unicode, and how it is accommodated by Unicode and non-Unicode ODBC drivers.
- "Security best practices for ODBC applications" describes the security best practices you should employ when developing and deploying your application with the driver.

Troubleshooting

The *Progress DataDirect for ODBC Drivers Reference* provides information on troubleshooting problems should they occur.

Refer to the "Troubleshooting" section in the *Progress DataDirect for ODBC Drivers Reference* for details.

See also

Operation timeouts on page 80
Out-of-memory issues on page 81

Contacting Technical Support

Progress DataDirect offers a variety of options to meet your support needs. Please visit our Web site for more details and for contact information:

https://www.progress.com/support

The Progress DataDirect Web site provides the latest support information through our global service network. The SupportLink program provides access to support contact details, tools, patches, and valuable information, including a list of FAQs for each product. In addition, you can search our Knowledgebase for technical bulletins and other information.

When you contact us for assistance, please provide the following information:

- Your number or the serial number that corresponds to the product for which you are seeking support, or a case number if you have been provided one for your issue. If you do not have a SupportLink contract, the SupportLink representative assisting you will connect you with our Sales team.
- Your name, phone number, email address, and organization. For a first-time call, you may be asked for full information, including location.

- The Progress DataDirect product and the version that you are using.
- The type and version of the operating system where you have installed your product.
- Any database, database version, third-party software, or other environment information required to understand the problem.
- A brief description of the problem, including, but not limited to, any error messages you have received, what steps you followed prior to the initial occurrence of the problem, any trace logs capturing the issue, and so on. Depending on the complexity of the problem, you may be asked to submit an example or reproducible application so that the issue can be re-created.
- A description of what you have attempted to resolve the issue. If you have researched your issue on Web search engines, our Knowledgebase, or have tested additional configurations, applications, or other vendor products, you will want to carefully note everything you have already attempted.
- A simple assessment of how the severity of the issue is impacting your organization.

October 2020, Release 8.0.0 for the Progress DataDirect for ODBC for Apache Cassandra Driver, Version 0001

2

Getting started

This chapter provides basic information about configuring your driver immediately after installation and testing your connection.

Information that the driver needs to connect to a database is stored in a *data source*. The ODBC specification describes three types of data sources: user data sources, system data sources (not a valid type on UNIX/Linux), and file data sources. On Windows, user and system data sources are stored in the registry of the local computer. The difference is that only a specific user can access user data sources, whereas any user of the machine can access system data sources. On Windows, UNIX, and Linux, file data sources, which are simply text files, can be stored locally or on a network computer, and are accessible to other machines.

When you define and configure a data source, you store default connection values for the driver that are used each time you connect to a particular database. You can change these defaults by modifying the data source.

For details, see the following topics:

- Configuring and connecting on Windows
- Configuring and connecting on UNIX and Linux

Configuring and connecting on Windows

The following basic information enables you to configure a data source and test connect with a driver immediately after installation. On Windows, you can configure and modify data sources through the ODBC Administrator using a driver Setup dialog box. Default connection values are specified through the options on the tabs of the Setup dialog box and are stored either as a user or system data source in the Windows Registry, or as a file data source in a specified location.

Setting the library path environment variable (Windows)

Before you can use your driver, you must set the PATH environment variable to include the path of the `jvm.dll` file of your Java™ Virtual Machine (JVM).

Note: The installer program sets the PATH environment variable to include the path of the `jvm.dll` file by default.

Configuring a data source

To configure a data source:

1. From the Progress DataDirect program group, start the ODBC Administrator and click either the **User DSN**, **System DSN**, or **File DSN** tab to display a list of data sources.

   - **User DSN**: If you installed a default DataDirect ODBC user data source as part of the installation, select the appropriate data source name and click **Configure** to display the driver Setup dialog box.

   If you are configuring a new user data source, click **Add** to display a list of installed drivers. Select the appropriate driver and click **Finish** to display the driver Setup dialog box.

   - **System DSN**: To configure a new system data source, click **Add** to display a list of installed drivers. Select the appropriate driver and click **Finish** to display the driver Setup dialog box.
   - **File DSN**: To configure a new file data source, click **Add** to display a list of installed drivers. Select the driver and click **Advanced** to specify attributes; otherwise, click **Next** to proceed. Specify a name for the data source and click **Next**. Verify the data source information; then, click **Finish** to display the driver Setup dialog box.

   The General tab of the Setup dialog box appears by default.

   Note: The General tab displays only fields that are required for creating a data source. The fields on all other tabs are optional, unless noted otherwise in this book.

   ---

2. On the General tab, provide the following information; then, select the **Schema Map** tab.

   Data Source Name: Type a string that identifies this data source configuration, such as Accounting.

   Host Name: Type the URL of the interface to which you want to connect.

   Port Number: Type the port number of the server listener. The default is `9042`.

   Keyspace Name: Type the name of the keyspace to which you want to connect. The default is `system`.

3. On the Schema Map tab, provide the following information; then, click **Apply**:

   Schema Map: Type the name and location of the configuration file where the relational map of native data is written. The driver either creates or looks for this file when connecting to the database. For example, `C:\Users\Default\AppData\Local\Progress\DataDirect\Cassandra_Schema\MainServer.config`.

   The default value is:

application_data_folder\Local\Progress\DataDirect\Cassandra_Schema*host_name*.config

where:

application_data_folder

is the name and location of the application data folder.

host_name

is the value specified for the Host Name connection option.

See also

Schema Map on page 112

Testing the connection

To test the connection:

1. After you have configured the data source, you can click **Test Connect** on the Setup dialog box to attempt to connect to the data source using the connection options specified in the dialog box. The driver returns a message indicating success or failure. A logon dialog box appears as described in "Using a logon dialog box."
2. Supply the requested information in the logon dialog box and click **OK**. Note that the information you enter in the logon dialog box during a test connect is not saved.
   - If the driver can connect, it releases the connection and displays a `Connection Established` message. Click **OK**.
   - If the driver cannot connect because of an incorrect environment or connection value, it displays an appropriate error message. Click **OK**.
3. On the driver Setup dialog box, click **OK**. The values you have specified are saved and are the defaults used when you connect to the data source. You can change these defaults by using the previously described procedure to modify your data source. You can override these defaults by connecting to the data source using a connection string with alternate values. See "Using a connection string" for information about using connection strings.

See also

Using a logon dialog box on page 70
Using a connection string on page 68

Configuring and connecting on UNIX and Linux

The following basic information enables you to configure a data source and test connect with a driver immediately after installation. See "Configuring and connecting to data sources" for detailed information about configuring the UNIX and Linux environments and data sources.

Note: In the following examples, *xx* in a driver filename represents the driver level number.

See also

Configuring and connecting to data sources on page 46

Environment configuration

To configure the environment:

1. Check your permissions: You must log in as a user with full r/w/x permissions recursively on the entire product installation directory.
2. From your login shell, determine which shell you are running by executing:

   ```
   echo $SHELL
   ```

3. Run one of the following product setup scripts from the installation directory to set variables: `odbc.sh` or `odbc.csh`. For Korn, Bourne, and equivalent shells, execute `odbc.sh`. For a C shell, execute `odbc.csh`. After running the setup script, execute:

   ```
   env
   ```

   to verify that the *`installation_directory`*`/lib` directory has been added to your shared library path.

4. Set the ODBCINI environment variable. The variable must point to the path from the root directory to the system information file where your data source resides. The system information file can have any name, but the product is installed with a default file called `odbc.ini` in the product installation directory. For example, if you use an installation directory of `/opt/odbc` and the default system information file, from the Korn or Bourne shell, you would enter:

   ```
   ODBCINI=/opt/odbc/odbc.ini; export ODBCINI
   ```

   From the C shell, you would enter:

   ```
   setenv ODBCINI /opt/odbc/odbc.ini
   ```

Test loading the driver

The ivtestlib (32-bit drivers) and ddtestlib (64-bit drivers) test loading tools are provided to test load drivers and help diagnose configuration problems in the UNIX and Linux environments, such as environment variables not correctly set or missing database client components. This tool is installed in the `/bin` subdirectory in the product installation directory. It attempts to load a specified ODBC driver and prints out all available error information if the load fails.

For example, if the drivers are installed in `/opt/odbc/lib`, the following command attempts to load the 32-bit driver on Solaris, where *`xx`* represents the version number of the driver:

```
ivtestlib /opt/odbc/lib/ivcsndrxx.so
```

Note: On Solaris, AIX, and Linux, the full path to the driver does not have to be specified for the tool. The HP-UX version, however, requires the full path.

If the load is successful, the tool returns a success message along with the version string of the driver. If the driver cannot be loaded,the tool returns an error message explaining why.

Setting the library path environment variable (UNIX and Linux)

Before you can use the driver for Apache Cassandra, you must set the library path environment variable for your UNIX/Linux operating system to include the directory containing your JVM's `libjvm.so` [`sl` | `a`] file, and that directory's parent directory.

NOTE FOR HP-UX: You also must set the LD_PRELOAD environment variable to the fully qualified path of the `libjvm.so`.

32-bit Driver: Library Path Environment Variable

Set the library path environment variable to include the directory containing your 32-bit JVM's `libjvm.so` [`sl` | `a`] file, and that directory's parent directory.

- `LD_LIBRARY_PATH` on Solaris, Linux, and HP-UX (Itanium)
- `SHLIB_PATH` on HP PA-RISC
- `LIBPATH` on AIX

64-bit Driver: Library Path Environment Variable

Set the library path environment variable to include the directory containing your 64-bit JVM's `libjvm.so` [`sl` | `a`] file, and that directory's parent directory.

- `LD_LIBRARY_PATH` on Solaris, HP-UX (Itanium), and Linux
- `LIBPATH` on AIX

Configuring a data source in the system information File

The default `odbc.ini` file installed in the installation directory is a template in which you create data source definitions on UNIX and Linux platforms. You enter your site-specific database connection information using a text editor. Each data source definition must include the keyword `Driver=`, which is the full path to the driver.

The following examples show the minimum connection string options that must be set to complete a test connection, where *xx* represents `iv` for 32-bit or `dd` for 64-bit drivers, and *yy* represents the extension. The values for the options are samples and are not necessarily the ones you would use.

```
[ODBC Data Sources]
Apache Cassandra=DataDirect 8.0 Apache Cassandra

[Apache Cassandra]
Driver=ODBCHOME/lib/xxcsndr28.yy
...
HostName=CassandraServer
...
KeyspaceName=CassandraKeyspace2
...
LogonID=John
...
Password=S3cr3t
...
PortNumber=9042
..
SchemaMap=/home/users/jsmith/Progress/DataDirect/Cassandra_Schema/CassandraServer.config
...
```

Connection Option Descriptions:

`HostName`: Either the name or the IP address of the server to which you want to connect.

KeySpaceName: The name of the keyspace to which you want to connect. The default is system.

LogonID: The default user ID that is used to connect to your database. Your ODBC application may override this value or you may override it in the logon dialog box or connection string.

Password: Specifies the password to use to connect to your database.

PortNumber: The port number of the server listener. The default is 9042.

SchemaMap: The name and location of the configuration file where the relational map of native data is written. The driver either creates or looks for this file when connecting to the database. The default is:

users_home_directory/Progress/DataDirect/Cassandra_Schema/*host_name*.config

where:

users_home_directory

is the user's home directory.

host_name

is the value specified for the HostName connection option.

See "Schema Map" for details.

See also

Schema Map on page 112

Testing the connection

The driver installation includes an ODBC application called example that can be used to connect to a data source and execute SQL. The application is located in the *installation_directory*/samples/example directory.

To run the program after setting up a data source in the odbc.ini, enter example and follow the prompts to enter your data source name, user name, and password. If successful, a SQL> prompt appears and you can type in SQL statements such as SELECT * FROM *table*. If example is unable to connect, the appropriate error message is returned.

3

Tutorials

The following sections guide you through using the driver to access your data with some common third-party applications:

- Accessing data in Tableau (Windows only) on page 37
- Accessing data in Microsoft Excel from the Data Connection Wizard (Windows only) on page 39
- Accessing data in Microsoft Excel from the Query Wizard (Windows only) on page 42

For details, see the following topics:

- The Example application
- Accessing data in Tableau (Windows only)
- Accessing data in Microsoft Excel from the Data Connection Wizard (Windows only)
- Accessing data in Microsoft Excel from the Query Wizard (Windows only)

The Example application

The driver installation includes an ODBC application called Example that can be used for:

- Testing any type of SQL statement
- Testing database connections
- Verifying your database environment

It can also be used to demonstrate ODBC function calls, including the following:

- SQLAllocHandle
- SQLBindCol
- SQLBindParameter
- SQLColAttribute
- SQLConnect
- SQLDescribeCol
- SQLDescribeParam
- SQLDisconnect
- SQLDriverConnect
- SQLExecDirect
- SQLFetch
- SQLFreeHandle
- SQLFreeStmt
- SQLGetDiagRec
- SQLGetInfo
- SQLNumResultCols
- SQLPrepare
- SQLSetEnvAttr
- SQLSetStmtAttr

The Example application can be built using the files located in the `\samples\examples` directory of the DataDirect for ODBC Drivers installation directory.

Note:

- For Windows, you can build the Windows app for ANSI and Unicode.
- For UNIX/Linux, instructions for building the Example application are contained inside the file `example.mak`, which can be read with a text editor.

To use the Example application:

1. After you have configured the data source, navigate to the `instal_dir\samples\example` directory.
2. Open the application using one of the following methods:
   - Running the application executable or binary:
     - On Windows, double-click the `Example.exe` file.
     - On UNIX/Linux, run the `example` application.
   - Executing a command-line argument. For example:
     - `example` *`connection_string`*
     - `example "DSN" "UID" "PWD"`
     - `example` *`connection_string`* `"`*`sql_command_1`*`" ["`*`sql_command_2`*`" ...]`

   Results: A command prompt opens.
3. Follow the prompts to enter your data source name, user name, and password. If successful, a SQL> prompt appears.
4. At the prompt, enter SQL statements to test your connection. For example:

   ```
   SELECT * FROM INFORMATION_SCHEMA.TABLES
   ```

The results of your query are displayed. If example is unable to connect, the appropriate error message is returned.

Accessing data in Tableau (Windows only)

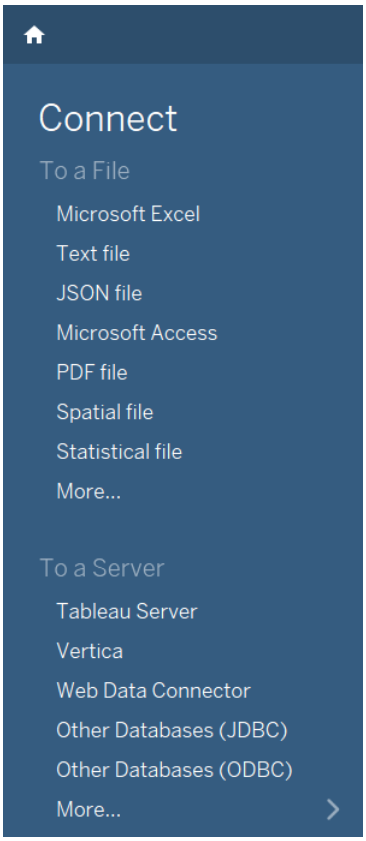
After you have configured your data source, you can use the driver to access your Cassandra data with Tableau. Tableau is a business intelligence software program that allows you to easily create reports and visualized representations of your data. By using the driver with Tableau, you can improve performance when retrieving data while leveraging the driver's relational mapping tools.

To use the driver to access data with Tableau:

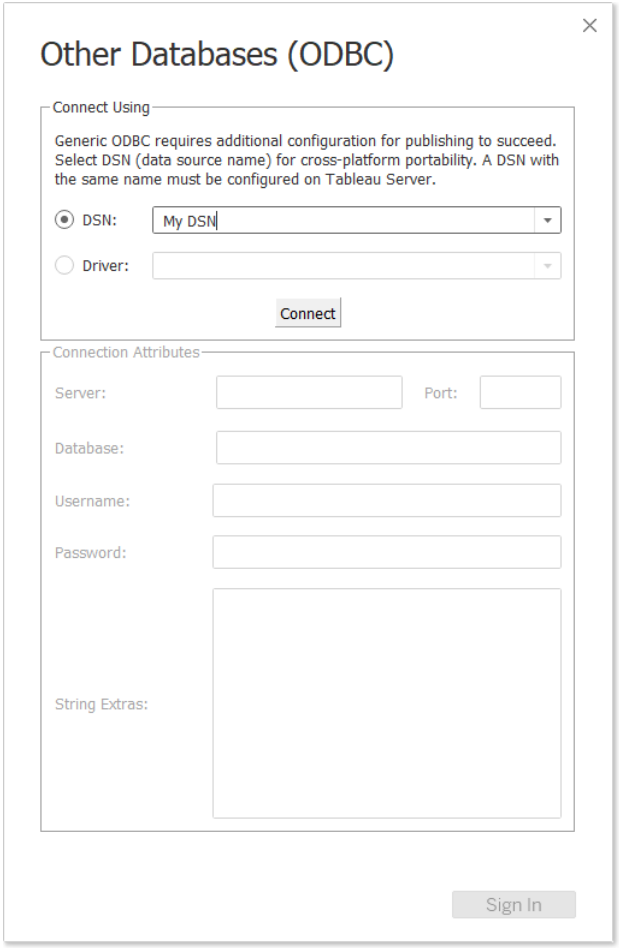
1. Navigate to the `\tools\Tableau` subdirectory of the Progress DataDirect installation directory; then, locate the Tableau data source file, `DataDirect Apache Cassandra.tdc`.
2. Copy the `DataDirect Apache Cassandra.tdc` into the following directory:

   `C:\Users\`*`user_name`*`\Documents\My Tableau Repository\Datasources`

3. Open Tableau. If the **Connect** menu does not open by default, select **Data > New Data Source** or the Add New Data Source button to open the menu.

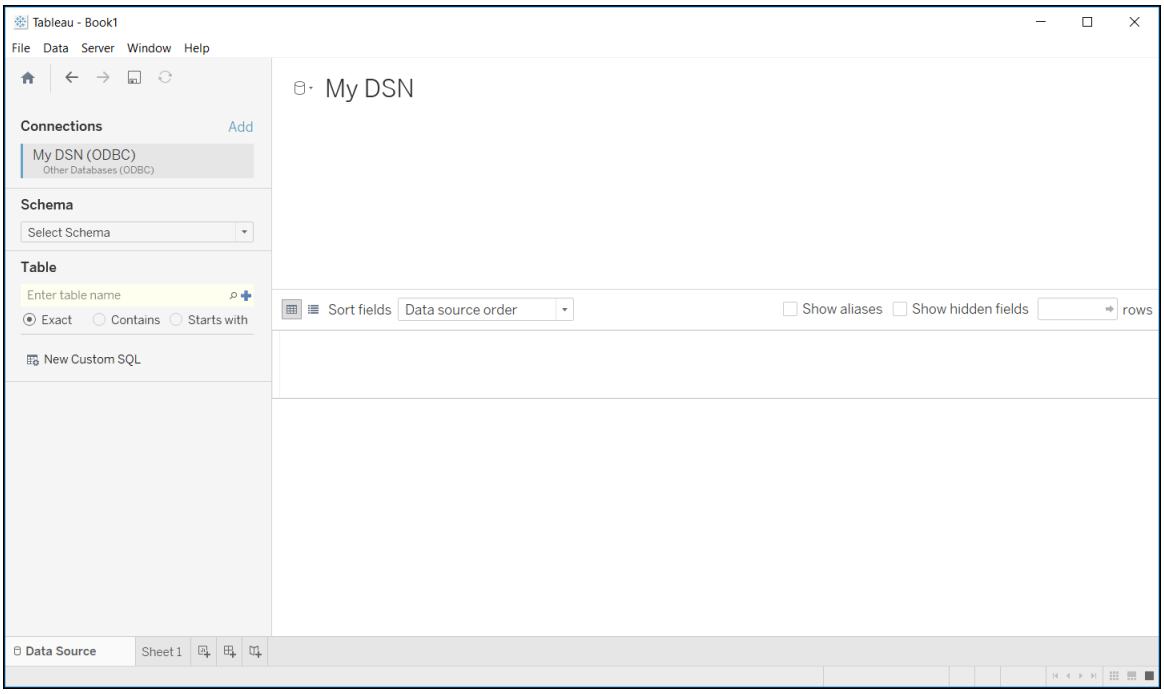


4. From the **Connect** menu, select **Other Databases (ODBC)**.
5. The **Server Connection** dialog appears.



In the DSN field, select the data source you want to use from the drop down menu. For example, **My DSN**. Then, click **Connect**. The **Logon to Cassandra** dialog appears pre-populated with the connection information you provided in your data source.

6. If required, type your user name and password; then, click **OK**. The Logon dialog closes. Then, click **OK** on the Server Connection dialog.
7. The **Data Source** window appears.



By default, Tableau connects live, or directly, to your data. We recommend that you use the default settings to avoid extracting all of your data. However, if you prefer, you can import your data by selecting the **Extract** option at the top of the dialog.

8. In the Schema field, select the database you want to use. The tables stored in this database are now available for selection in the Table field.

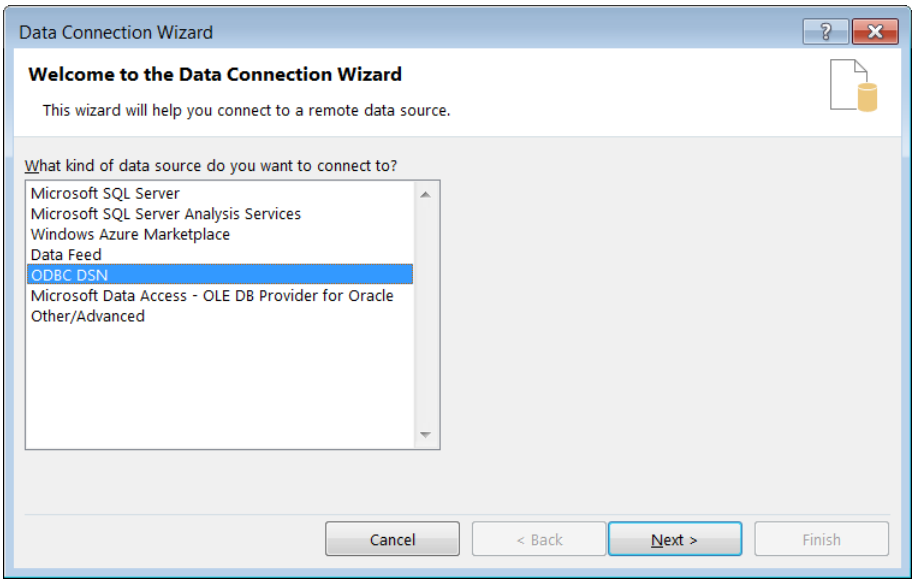
You have successfully accessed your data and are now ready to create reports with Tableau. For more information, refer to the Tableau product documentation at: http://www.tableau.com/support/help.

Accessing data in Microsoft Excel from the Data Connection Wizard (Windows only)

After you have configured your data source, you can use the driver to access your data with Microsoft Excel from the Data Connection Wizard. Using the driver with Excel provides improved performance when retrieving data, while leveraging the driver's relational-mapping tools.

To use the driver to access data with Excel from the Data Connection Wizard:

1. Open your workbook in Excel.
2. From the **Data** menu, select **From Other Sources**>**From Data Connection Wizard**.
3. The **Welcome to Data Connection Wizard** dialog appears.



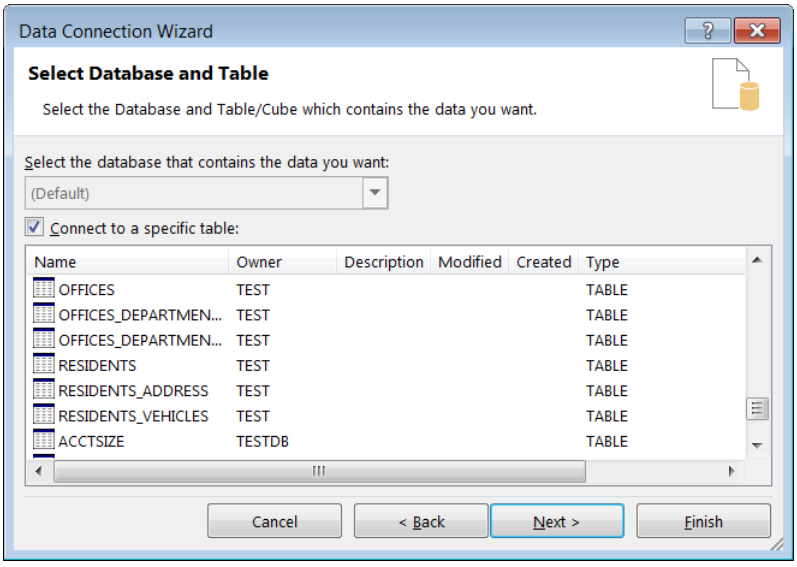
Select **ODBC DSN** from the data source list; then, click **Next**.

4. From the ODBC data sources list, select your data source. Click **Next**.
5. The logon dialog appears pre-populated with the connection information you provided in your data source. If required, type your password. Click **OK** to proceed.

Note: The logon dialog may reappear if Excel needs to access additional information from the data source. If this occurs, re-enter your password; then, click **OK** to proceed to the next step.

---

6. The **Select Database and Table** window appears.

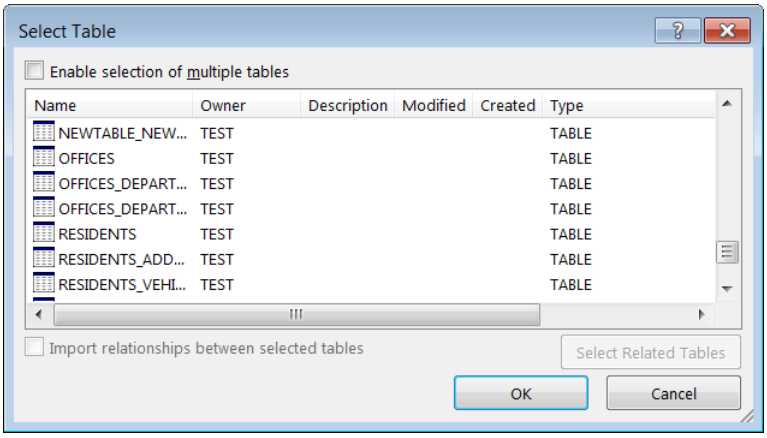


- To access data from multiple tables, uncheck the **Connect to a specific table** box; then, click **Next**.
- To access data from a specific table, select the table you want to import from the list; then, click **Next**.

7. Enter the file name, description, and, optionally, the friendly name and search keywords for your Data Connection file in the corresponding fields. Click **Finish**.

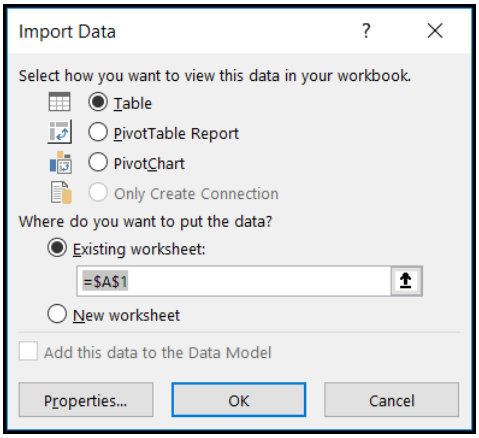
   - If you chose to access data from multiple tables in Step 6 on page 40, proceed to the next step.
   - If you chose to access data from a specific table in Step 6 on page 40, skip to step 9 on page 41.

8. The Select Table window appears.



   Select the tables that you want to import into your workbook; then, click **OK**.

9. The Import Data window appears.



   Select the desired view and insertion point for the data. Click **OK**.

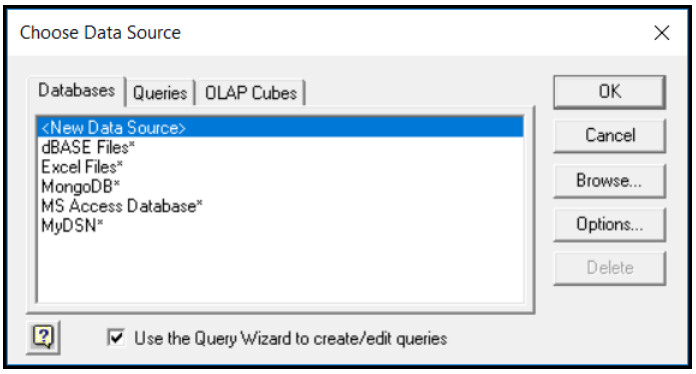
You have successfully accessed your data in Excel using the Data Connection Wizard. For more information, refer to the Microsoft Excel product documentation at: https://support.office.com/.

Accessing data in Microsoft Excel from the Query Wizard (Windows only)

After you have configured your data source, you can use the driver to access your data with Microsoft Excel from the Query Wizard. Using the driver with Excel provides improved performance when retrieving data, while leveraging the driver's relational-mapping tools.

To use the driver to access data with Excel from the Query Wizard:

1. Open your workbook in Excel.
2. From the **Data** menu, select **Get Data**>**From Other Sources**>**From Microsoft Query**.
3. The **Choose Data Source** dialog appears.



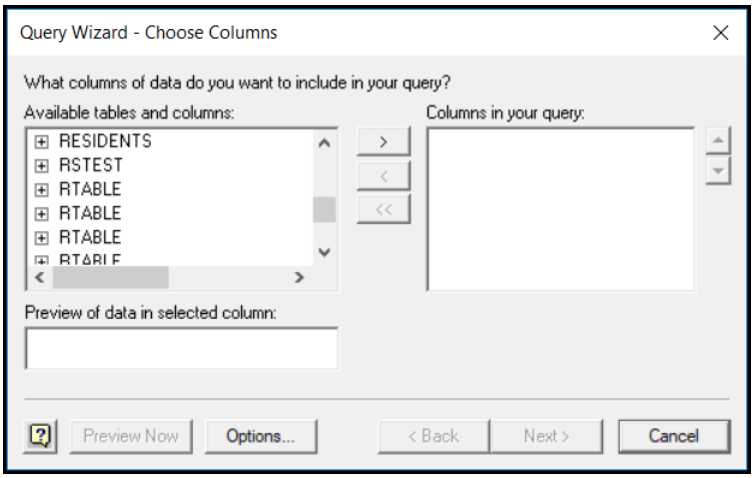
From the Databases list, select your data source. For example, **MyDSN**. Click **OK**.

4. The logon dialog appears pre-populated with the connection information you provided in your data source. If required, type your password. Click **OK** to proceed.

Note: The logon dialog may reappear if Excel needs to access additional information from the data source. If this occurs, re-enter your password; then, click **OK** to proceed to the next step.

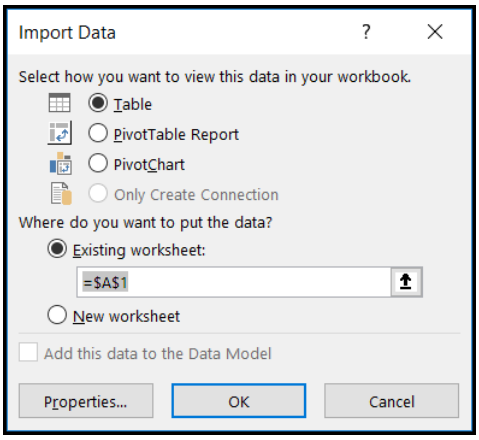
---

5. The **Query Wizard - Choose Columns** window appears.



Choose the columns you want to import into your workbook. To add a column, select the column name in Available tables and columns pane; then, click the **>** button. After you add the columns you want to include, click **Next** to continue.

6. Optionally, filter your data using the drop-down menus; then, click **Next**.
7. Optionally, sort your data using the drop-down menus; then, click **Next**.
8. Select "Return Data to Microsoft Excel"; then, click **Finish**.
9. The **Import Data** window appears.



Select the desired view and insertion point for your data. Click **OK**.

You have successfully accessed your data in Excel using the Query Wizard. For more information, refer to the Microsoft Excel product documentation at: https://support.office.com/.

4

Using the driver

This section guides you through the configuring and connecting to data sources. In addition, it explains how to use the functionality supported by your driver.

For details, see the following topics:

Configuring and connecting to data sources

After you install the driver, you configure data sources to connect to the database. See "Getting started" for an explanation of different types of data sources. The data source contains connection options that allow you to tune the driver for specific performance. If you want to use a data source but need to change some of its values, you can either modify the data source or override its values at connection time through a connection string.

If you choose to use a connection string, you must use specific connection string attributes. See "Using a connection string" for an alphabetical list of driver connection string attributes and their initial default values.

See also

Getting started on page 29
Using a connection string on page 68

Configuring the product on UNIX/Linux

UNIX®

This chapter contains specific information about using your driver in the UNIX and Linux environments.

See "Environment variables" for additional platform information.

See also

Environment variables on page 46

Environment variables

The first step in setting up and configuring the driver for use is to set several environment variables. The following procedures require that you have the appropriate permissions to modify your environment and to read, write, and execute various files. You must log in as a user with full r/w/x permissions recursively on the entire Progress DataDirect *for* ODBC installation directory.

Library search path

The library search path variable can be set by executing the appropriate shell script located in the ODBC home directory. From your login shell, determine which shell you are running by executing:

```
echo $SHELL
```

C shell login (and related shell) users must execute the following command before attempting to use ODBC-enabled applications:

```
source ./odbc.csh
```

Bourne shell login (and related shell) users must initialize their environment as follows:

```
. ./odbc.sh
```

Executing these scripts sets the appropriate library search path environment variable:

- `LD_LIBRARY_PATH` on HP-UX IPF, Linux, and Oracle Solaris
- `LIBPATH` on AIX

- SHLIB_PATH on HP-UX PA-RISC

The library search path environment variable must be set so that the ODBC core components and drivers can be located at the time of execution. After running the setup script, execute:

```
env
```

to verify that the *installation_directory*/lib directory has been added to your shared library path.

ODBCINI

Setup installs in the product installation directory a default system information file, named odbc.ini, that contains data sources. See "Data source configuration on UNIX/Linux" for an explanation of the odbc.ini file. The system administrator can choose to rename the file and/or move it to another location. In either case, the environment variable ODBCINI must be set to point to the fully qualified path name of the odbc.ini file.

For example, to point to the location of the file for an installation on /opt/odbc in the C shell, you would set this variable as follows:

```
setenv ODBCINI /opt/odbc/odbc.ini
```

In the Bourne or Korn shell, you would set it as:

```
ODBCINI=/opt/odbc/odbc.ini;export ODBCINI
```

As an alternative, you can choose to make the odbc.ini file a hidden file and not set the ODBCINI variable. In this case, you would need to rename the file to .odbc.ini (to make it a hidden file) and move it to the user's $HOME directory.

The driver searches for the location of the odbc.ini file as follows:

1. The driver checks the ODBCINI variable
2. The driver checks $HOME for .odbc.ini

If the driver does not locate the system information file, it returns an error.

See also

Data source configuration on UNIX/Linux on page 49

ODBCINST

Setup installs in the product installation directory a default file, named odbcinst.ini, for use with DSN-less connections. See "DSN-less connections" for an explanation of the odbcinst.ini file. The system administrator can choose to rename the file or move it to another location. In either case, the environment variable ODBCINST must be set to point to the fully qualified path name of the odbcinst.ini file.

For example, to point to the location of the file for an installation on /opt/odbc in the C shell, you would set this variable as follows:

```
setenv ODBCINST /opt/odbc/odbcinst.ini
```

In the Bourne or Korn shell, you would set it as:

```
ODBCINST=/opt/odbc/odbcinst.ini;export ODBCINST
```

As an alternative, you can choose to make the odbcinst.ini file a hidden file and not set the ODBCINST variable. In this case, you would need to rename the file to .odbcinst.ini (to make it a hidden file) and move it to the user's $HOME directory.

The driver searches for the location of the odbcinst.ini file as follows:

1. The driver checks the ODBCINST variable
2. The driver checks $HOME for .odbcinst.ini

If the driver does not locate the odbcinst.ini file, it returns an error.

See also

DSN-less connections on page 52

DD_INSTALLDIR

This variable provides the driver with the location of the product installation directory so that it can access support files. DD_INSTALLDIR must be set to point to the fully qualified path name of the installation directory.

For example, to point to the location of the directory for an installation on /opt/odbc in the C shell, you would set this variable as follows:

```
setenv DD_INSTALLDIR /opt/odbc
```

In the Bourne or Korn shell, you would set it as:

```
DD_INSTALLDIR=/opt/odbc;export DD_INSTALLDIR
```

The driver searches for the location of the installation directory as follows:

1. The driver checks the DD_INSTALLDIR variable
2. The driver checks the odbc.ini or the odbcinst.ini files for the InstallDir keyword (see "Configuration through the system information (odbc.ini) file" for a description of the InstallDir keyword)

If the driver does not locate the installation directory, it returns an error.

The next step is to test load the driver.

See also

Configuration through the system information (odbc.ini) file on page 49

The test loading tool

The second step in preparing to use a driver is to test load it.

The ivtestlib (32-bit driver) and ddtestlib (64-bit driver) test loading tools are provided to test load drivers and help diagnose configuration problems in the UNIX and Linux environments, such as environment variables not correctly set or missing database client components. This tool is installed in the /bin subdirectory in the product installation directory. It attempts to load a specified ODBC driver and prints out all available error information if the load fails.

The test loading tool is provided to test load drivers and help diagnose configuration problems in the UNIX and Linux environments, such as environment variables not correctly set or missing database client components. This tool is installed in the bin subdirectory in the product installation directory. It attempts to load a specified ODBC driver and prints out all available error information if the load fails.

For example, if the driver is installed in /opt/odbc/lib, the following command attempts to load the 32-bit driver on Solaris, where *xx* represents the version number of the driver:

```
ivtestlib /opt/odbc/lib/ivcsndrxx.so
```

Note: On Solaris, AIX, and Linux, the full path to the driver does not have to be specified for the tool. The HP-UX version, however, requires the full path.

If the load is successful, the tool returns a success message along with the version string of the driver. If the driver cannot be loaded, the tool returns an error message explaining why.

See "Version string information" for details about version strings.

The next step is to configure a data source through the system information file.

See also

Version string information on page 14

Data source configuration on UNIX/Linux

In the UNIX and Linux environments, a system information file is used to store data source information. Setup installs a default version of this file, called `odbc.ini`, in the product installation directory. This is a plain text file that contains data source definitions.

Configuration through the system information (odbc.ini) file

To configure a data source manually, you edit the `odbc.ini` file with a text editor. The content of this file is divided into three sections.

At the beginning of the file is a section named `[ODBC Data Sources]` containing *`data_source_name=installed-driver`* pairs, for example:

```
Apache Cassandra=DataDirect 8.0 Apache Cassandra Driver.
```

The driver uses this section to match a data source to the appropriate installed driver.

The `[ODBC Data Sources]` section also includes data source definitions. The default `odbc.ini` contains a data source definition for the driver. Each data source definition begins with a data source name in square brackets, for example, `[Apache Cassandra]`. The data source definitions contain connection string *attribute=value* pairs with default values. You can modify these values as appropriate for your system. "Connection Option Descriptions" describes these attributes. See "Sample default odbc.ini file" for sample data sources.

The second section of the file is named `[ODBC File DSN]` and includes one keyword:

```
[ODBC File DSN]
DefaultDSNDir=
```

This keyword defines the path of the default location for file data sources (see "File data sources").

Note: This section is not included in the default `odbc.ini` file that is installed by the product installer. You must add this section manually.

The third section of the file is named `[ODBC]` and includes several keywords, for example:

```
[ODBC]
IANAAppCodePage=4
InstallDir=/opt/odbc
Trace=0
TraceFile=odbctrace.out
TraceDll=/opt/odbc/lib/ivtrc28.so
ODBCTraceMaxFileSize=102400
ODBCTraceMaxNumFiles=10
```

The IANAAppCodePage keyword defines the default value that the UNIX/Linux driver uses if individual data sources have not specified a different value. See "IANAAAppCodePage" in "Connection option descriptions". The default value is `4`.

For supported code page values, refer to "Code page values" in the *Progress DataDirect for ODBC Drivers Reference*.

The `InstallDir` keyword must be included in this section. The value of this keyword is the path to the installation directory under which the `/lib` and `/locale` directories are contained. The installation process automatically writes your installation directory to the default `odbc.ini` file.

For example, if you choose an installation location of `/opt/odbc`, then the following line is written to the `[ODBC]` section of the default `odbc.ini`:

```
InstallDir=/opt/odbc
```

Note: If you are using only DSN-less connections through an `odbcinst.ini` file and do not have an `odbc.ini` file, then you must provide `[ODBC]` section information in the `[ODBC]` section of the `odbcinst.ini` file. The driver and Driver Manager always check first in the `[ODBC]` section of an `odbc.ini` file. If no `odbc.ini` file exists or if the `odbc.ini` file does not contain an `[ODBC]` section, they check for an `[ODBC]` section in the `odbcinst.ini` file. See "DSN-less connections" for details.

ODBC tracing allows you to trace calls to the ODBC driver and create a log of the traces for troubleshooting purposes. The following keywords all control tracing: `Trace`, `TraceFile`, `TraceDLL`, `ODBCTraceMaxFileSize`, and `ODBCTraceMaxNumFiles`.

For a complete discussion of tracing, refer to "ODBC trace" in the *Progress DataDirect for ODBC Drivers Reference*.

See also

Connection option descriptions on page 87
Sample default odbc.ini file on page 50
File data sources on page 53
IANAAppCodePage on page 100
DSN-less connections on page 52

Sample default odbc.ini file

The following is a sample `odbc.ini` file that the installer program installs in the installation directory. All occurrences of ODBCHOME are replaced with your installation directory path during installation of the file. Values that you must supply are enclosed by angle brackets (< >). If you are using the installed `odbc.ini` file, you must supply the values and remove the angle brackets before that data source section will operate properly. Commented lines are denoted by the # symbol. This sample shows a 32-bit driver with the driver file name beginning with iv. A 64-bit driver file would be identical except that driver name would begin with dd and the list of data sources would include only the 64-bit drivers.

```
[ODBC Data Sources]
Apache Cassandra=DataDirect 8.0 Apache Cassandra

[Apache Cassandra]
Driver=ODBCHOME/lib/ivcsndr28.so
Description=DataDirect 8.0 Apache Cassandra
ApplicationUsingThreads=1
AsciiSize=4000
AuthenticationMethod=0
ClusterNodes=
ConfigOptions=
```

```
CreateMap=2
DataSourceName=
DecimalPrecision=38
DecimalScale=10
FetchSize=100
HostName=
InitializationString=
JVMArgs=-Xmx256m
JVMClasspath=
KeySpaceName=
LogConfigFile=
LoginTimeout=15
LogonID=
NativeFetchSize=10000
Password=
PortNumber=9042
ReadConsistency=4
ReadOnly=0
ReportCodepageConversionErrors=0
ResultMemorySize=-1
SchemaMap=
ServerPortNumber=19934
SQLEngineMode=0
TransactionMode=0
VarcharSize=4000
VarintPrecision=38
WriteConsistency=4

[ODBC]
IANAAppCodePage=4
InstallDir=ODBCHOME
Trace=0
TraceFile=odbctrace.out
TraceDll=ODBCHOME/lib/ivtrc28.so
ODBCTraceMaxFileSize=102400
ODBCTraceMaxNumFiles=10

[ODBC File DSN]
DefaultDSNDir=
UseCursorLib=0
```

To modify or create data sources in the `odbc.ini` file, use the following procedures.

- **To modify a data source:**
  a) Using a text editor, open the `odbc.ini` file.
  b) Modify the default attributes in the data source definitions as necessary based on your system specifics, for example, enter the host name and port number of your system in the appropriate location.

     Consult the "Apache Cassandra Attribute Names" table in "Connection option descriptions" for other specific attribute values.

  c) After making all modifications, save the `odbc.ini` file and close the text editor.

  Important: The "Apache Cassandra Attribute Names" table in "Connection option descriptions" lists both the long and short names of the attribute. When entering attribute names into `odbc.ini`, you must use the long name of the attribute. The short name is not valid in the `odbc.ini` file.

- **To create a new data source:**
  a) Using a text editor, open the `odbc.ini` file.
  b) Copy an appropriate existing default data source definition and paste it to another location in the file.

c) Change the data source name in the copied data source definition to a new name. The data source name is between square brackets at the beginning of the definition, for example, `[Apache Cassandra]`.

d) Modify the attributes in the new definition as necessary based on your system specifics, for example, enter the host name and port number of your system in the appropriate location.

   Consult the "Apache Cassandra Attribute Names" table in "Connection option descriptions" for other specific attribute values.

e) In the `[ODBC]` section at the beginning of the file, add a new *`data_source_name=installed-driver`* pair containing the new data source name and the appropriate installed driver name.

f) After making all modifications, save the `odbc.ini` file and close the text editor.

Important: The "Apache Cassandra Attribute Names" table in "Connection option descriptions" lists both the long and short name of the attribute. When entering attribute names into `odbc.ini`, you must use the long name of the attribute. The short name is not valid in the `odbc.ini` file.

See also

Connection option descriptions on page 87

DSN-less connections

Connections to a data source can be made via a connection string without referring to a data source name (DSN-less connections). This is done by specifying the `DRIVER=` keyword instead of the `DSN=` keyword in a connection string, as outlined in the ODBC specification. A file named `odbcinst.ini` must exist when the driver encounters `DRIVER=` in a connection string.

Setup installs a default version of this file in the product installation directory (see "ODBCINST" for details about relocating and renaming this file). This is a plain text file that contains default DSN-less connection information. You should not normally need to edit this file. The content of this file is divided into several sections.

At the beginning of the file is a section named `[ODBC Drivers]` that lists installed drivers, for example,

```
DataDirect 8.0 Apache Cassandra Driver=Installed
```

This section also includes additional information for each driver.

The next section of the file is named `[Administrator]`. The keyword in this section, `AdminHelpRootDirectory`, is required for the Linux ODBC Administrator to locate its help system. The installation process automatically provides the correct value for this keyword.

The final section of the file is named `[ODBC]`. The `[ODBC]` section in the `odbcinst.ini` file fulfills the same purpose in DSN-less connections as the `[ODBC]` section in the odbc.ini file does for data source connections. See "Configuration through the system information (odbc.ini) file" for a description of the other keywords this section.

Note: The `odbcinst.ini` file and the `odbc.ini` file include an `[ODBC]` section. If the information in these two sections is not the same, the values in the `odbc.ini` `[ODBC]` section override those of the `odbcinst.ini` `[ODBC]` section.

See also

ODBCINST on page 47
Configuration through the system information (odbc.ini) file on page 49

Sample odbcinst.ini file

The following is a sample `odbcinst.ini`. All occurrences of `ODBCHOME` are replaced with your installation directory path during installation of the file. Commented lines are denoted by the # symbol. This sample shows a 32-bit driver with the driver file name beginning with `iv`; a 64-bit driver file would be identical except that driver names would begin with `dd`.

```
[ODBC Drivers]
DataDirect 8.0 Apache Cassandra=Installed

[DataDirect 8.0 Apache Cassandra]
Driver=ODBCHOME/lib/ivcsndr28.so
JarFile=ODBCHOME/java/lib/cassandra.jar
APILevel=1
ConnectFunctions=YYY
CPTimeout=60
DriverODBCVer=3.52
FileUsage=0
HelpRootDirectory=ODBCHOME/CassandraHelp
Setup=
SQLLevel=0
UsageCount=1

[ODBC]
#This section must contain values for DSN-less connections
#if no odbc.ini file exists. If an odbc.ini file exists,
#the values from that [ODBC] section are used.

IANAAppCodePage=4
InstallDir=ODBCHOME
Trace=0
TraceFile=odbctrace.out
TraceDll=ODBCHOME/lib/ivtrc28.so
ODBCTraceMaxFileSize=102400
ODBCTraceMaxNumFiles=10
```

File data sources

The Driver Manager on UNIX and Linux supports file data sources. The advantage of a file data source is that it can be stored on a server and accessed by other machines, either Windows, UNIX, or Linux. See "Getting started" for a general description of ODBC data sources on both Windows and UNIX.

A file data source is simply a text file that contains connection information. It can be created with a text editor. The file normally has an extension of `.dsn`.

For example, a file data source for the driver would be similar to the following:

```
[ODBC]
Driver=DataDirect 8.0 Apache Cassandra
Port=9042
HostName=Cassandra2
KeyspaceName=CassandraKeyspace2
SchemaMap=/home/users/jsmith/Progress/DataDirect/Cassandra_Schema/Cassandra2.config
LogonID=jsmith
```

It must contain all basic connection information plus any optional attributes. Because it uses the `DRIVER=` keyword, an `odbcinst.ini` file containing the driver location must exist (see "DSN-less connections").

The file data source is accessed by specifying the `FILEDSN=` instead of the `DSN=` keyword in a connection string, as outlined in the ODBC specification. The complete path to the file data source can be specified in the syntax that is normal for the machine on which the file is located. For example, on Windows:

```
FILEDSN=C:\Program Files\Common Files\ODBC\DataSources\Cassandra2.dsn
```

or, on UNIX and Linux:

```
FILEDSN=/home/users/jsmith/filedsn/Cassandra2.dsn
```

If no path is specified for the file data source, the Driver Manager uses the DefaultDSNDir property, which is defined in the `[ODBC File DSN]` setting in the `odbc.ini` file to locate file data sources (see "Data source configuration on UNIX/Linux" for details). If the `[ODBC File DSN]` setting is not defined, the Driver Manager uses the InstallDir setting in the [ODBC] section of the `odbc.ini` file. The Driver Manager does not support the SQLReadFileDSN and SQLWriteFileDSN functions.

As with any connection string, you can specify attributes to override the default values in the data source:

```
FILEDSN=/home/users/jsmith/filedsn/Cassandra2.dsn;UID=jsmith;PWD=test01
```

See also

Getting started on page 29
DSN-less connections on page 52
Data source configuration on UNIX/Linux on page 49

UTF-16 applications on UNIX and Linux

Because the DataDirect Driver Manager allows applications to use either UTF-8 or UTF-16 Unicode encoding, applications written in UTF-16 for Windows platforms can also be used on UNIX and Linux platforms.

The Driver Manager assumes a default of UTF-8 applications; therefore, two things must occur for it to determine that the application is UTF-16:

- The definition of SQLWCHAR in the ODBC header files must be switched from "char *" to "short *". To do this, the application uses #define SQLWCHARSHORT.
- The application must set the encoding for the environment or connection using one of the following attributes. If your application passes UTF-8 encoded strings to some connections and UTF-16 encoded strings to other connections in the same environment, encoding should be set for the connection only; otherwise, either method can be used.
  - To configure the encoding for the environment, set the ODBC environment attribute SQL_ATTR_APP_UNICODE_TYPE to a value of `SQL_DD_CP_UTF16`, for example:

    ```
    rc = SQLSetEnvAttr(*henv,
    SQL_ATTR_APP_UNICODE_TYPE,(SQLPOINTER)SQL_DD_CP_UTF16, SQL_IS_INTEGER);
    ```

  - To configure the encoding for the connection only, set the ODBC connection attribute SQL_ATTR_APP_UNICODE_TYPE to a value of SQL_DD_CP_UTF16. For example:

    ```
    rc = SQLSetConnectAttr(hdbc, SQL_ATTR_APP_UNICODE_TYPE, SQL_DD_CP_UTF16,
    SQL_IS_INTEGER);
    ```

Data source configuration through a GUI

On Windows, data sources are stored in the Windows Registry. You can configure and modify data sources through the ODBC Administrator using a driver Setup dialog box, as described in this section.

When the driver is first installed, the values of its connection options are set by default. These values appear on the driver Setup dialog box tabs when you create a new data source. You can change these default values by modifying the data source. In the following procedure, the description of each tab is followed by a table that lists the connection options for that tab and their initial default values. This table links you to a complete description of the options and their connection string attribute equivalents. The connection string attributes are used to override the default values of the data source if you want to change these values at connection time.

To configure a Apache Cassandra data source:

1. Start the ODBC Administrator by selecting its icon from the Progress DataDirect for ODBC program group.
2. Select a tab:

   - **User DSN**: If you are configuring an existing user data source, select the data source name and click **Configure** to display the driver Setup dialog box.

     If you are configuring a new user data source, click **Add** to display a list of installed drivers. Select the driver and click **Finish** to display the driver Setup dialog box.

   - **System DSN**: If you are configuring an existing system data source, select the data source name and click **Configure** to display the driver Setup dialog box.

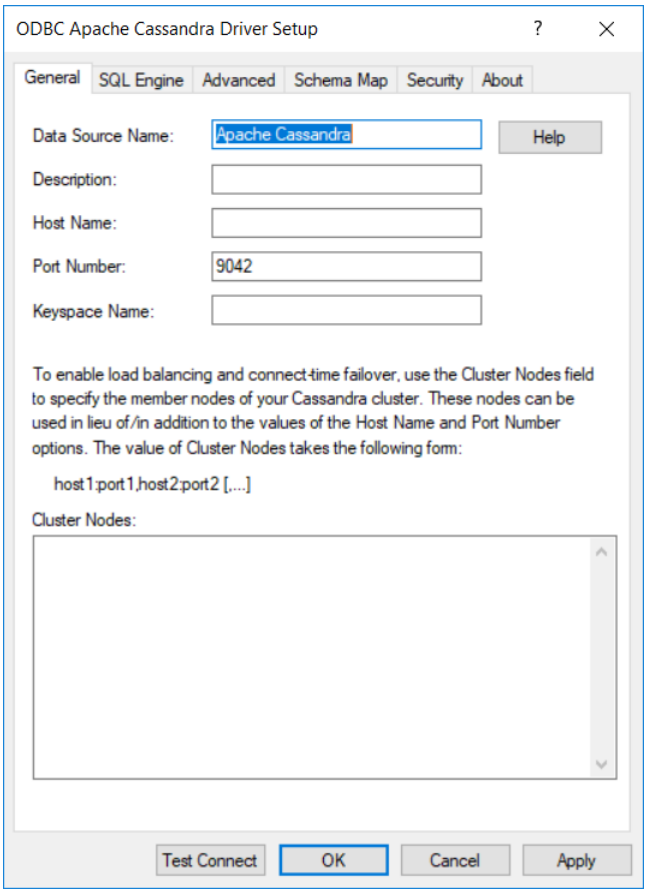
     If you are configuring a new system data source, click **Add** to display a list of installed drivers. Select the driver and click `Finish` to display the driver Setup dialog box.

   - **File DSN**: If you are configuring an existing file data source, select the data source file and click **Configure** to display the driver Setup dialog box.

     If you are configuring a new file data source, click **Add** to display a list of installed drivers; then, select a driver. Click **Advanced** if you want to specify attributes; otherwise, click **Next** to proceed. Specify a name for the data source and click **Next**. Verify the data source information; then, click **Finish** to display the driver Setup dialog box.

3. The General tab of the Setup dialog box appears by default.

Figure 1: General tab



On this tab, provide values for the options in the following table; then, click **Apply.** The table provides links to descriptions of the connection options. The General tab displays fields that are required for creating a data source. The fields on all other tabs are optional, unless noted otherwise.

Connection Options: General	Description
Data Source Name on page 96	Specifies the name of a data source in your Windows Registry or `odbc.ini` file. **Default:** None
Description on page 98	Specifies an optional long description of a data source. This description is not used as a runtime connection attribute, but does appear in the `ODBC.INI` section of the Registry and in the `odbc.ini` file. **Default:** None
Host Name on page 99	The name or the IP address of the server to which you want to connect. **Default:** None
Port Number on page 108	Specifies the port number of the server listener. **Default:** `9042`

Connection Options: General	Description
Keyspace Name on page 104	Specifies the name of the keyspace to which you want to connect. **Default:** `system`
Cluster Nodes on page 92	A comma-separated list of member nodes in your cluster to which the driver attempts to connect. Specifying a value for this option enables connect time failover and load balancing. **Default:** None

4. At any point during the configuration process, you can click **Test Connect** to attempt to connect to the data source using the connection options specified in the driver Setup dialog box. A logon dialog box appears (see "Using a logon dialog box" for details). Note that the information you enter in the logon dialog box during a test connect is not saved.
5. To further configure your driver, click on the following tabs. The corresponding sections provide details on the fields specific to each configuration tab:
   - SQL Engine tab allows you to configure the SQL engine's behavior.
   - Advanced tab allows you to configure advanced behavior.
   - Schema Map tab allows you to configure mapping behavior.
   - Security tab allows you to configure security settings.
6. Click **OK**. When you click **OK**, the values you have specified become the defaults when you connect to the data source. You can change these defaults by using this procedure to reconfigure your data source. You can override these defaults by connecting to the data source using a connection string with alternate values.

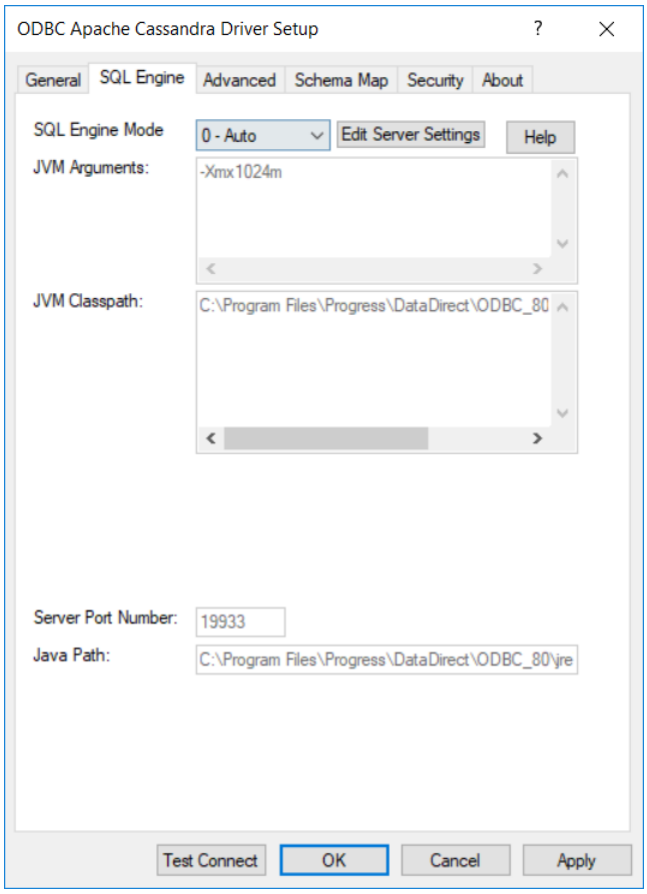
See also

Using a logon dialog box on page 70

SQL Engine tab

The SQL Engine tab allows you to specify additional data source settings. The fields are optional unless otherwise noted. On this tab, provide values for the options in the following tables; then, click **Apply**.

Figure 2: SQL Engine tab



The SQL engine can be run in one of two modes: direct mode or server mode. When set to direct mode, both the driver and its SQL engine run in the ODBC application's address space. Some applications may experience problems loading the JVM because the process exceeds the available heap space. To avoid this issue, you can configure the driver to operate in server mode. Server mode allows the driver to connect to an SQL engine JVM running as a separate service.

By default, the driver is set to **0 - Auto**. In this setting, the SQL engine attempts to run in server mode first, but will failover to direct mode if server mode is unavailable. If you prefer that the SQL engine runs exclusively in a particular mode, set the SQL Engine Mode option to **1 – Server** to run only in server mode or **2 – Direct** to run only in direct mode.

Table 12: SQL Engine tab Connection Options

Connection Options: SQL Engine	Default
SQL Engine Mode on page 114	If set to **0 - Auto**, the SQL engine attempts to run in server mode first; however, if server mode is unavailable, it runs in direct mode. If set to **1 - Server**, the SQL engine runs in server mode. The SQL engine operates in a separate process from the driver within its own JVM. If the SQL engine is unavailable, the connection will fail. If set to **2 - Direct**, the SQL engine runs in direct mode. The driver and its SQL engine run in a single process within the same JVM. **Important:** When the SQL engine is configured to run in server mode (**0-Auto** | **1-Server**), you must start the SQL engine service before using the driver (see "Starting the SQL engine server" for more information). Multiple drivers on different clients can use the same service. **Important:** Changes you make to the server mode configuration affect all DSNs sharing the service. **Default: 0 - Auto**
JVM Arguments on page 102	A string that contains the arguments that are passed to the JVM that the driver is starting. The location of the JVM must be specified on the driver library path. Values that include special characters or spaces must be enclosed in curly braces { } when used in a connection string. **Default:** For the 32-bit driver: `-Xmx256m` For the 64-bit driver: `-Xmx1024m`
JVM Classpath on page 103	Specifies the CLASSPATH for the Java Virtual Machine (JVM) used by the driver. The CLASSPATH is the search string the JVM uses to locate the Java jar files the driver needs. Separate multiple jar files by a semi-colon on Windows platforms and by a colon on all other platforms. CLASSPATH values with multiple jar files must be enclosed in curly braces { } when used in a connection string. **Note:** If no value is specified, the driver automatically detects the CLASSPATHs for all ODBC drivers installed on your machine. **Default:** Empty String

When set to **0 - Auto** or **1 – Server**, additional configuration settings that are specific to server mode are exposed in the setup dialog. The settings for server mode are read only in the Driver Setup Dialog. For a description of these settings, see the table below.

To define the settings for server mode, click **Edit Server Settings** from the SQL Engine tab. The SQL Engine Service Setup dialog box appears.

Caution: Modifying the Server Settings will affect all DSNs using this service.

Note: You must be an administrator to modify the server mode settings. Otherwise, the Edit Server Settings button does not appear on the SQL Engine tab.

You use the SQL Engine Service Setup dialog box to configure server mode and to start or stop the service. See "Configuring server mode" for detailed information.

Table 13: Server Mode Configuration Options

Configuration Options: SQL Engine Service	Description
Server Port Number on page 113	Specifies a valid port on which the SQL engine listens for requests from the driver. **Default:** For the 32-bit driver: `19934` For the 64-bit driver: `19933`
Java Path	Specifies fully qualified path to the JVM executable that you want to use to run the SQL engine server. The path must not contain double quotation marks. **Default:** The fully qualified path to the JVM executable (`java.exe`)

If you finished configuring your driver, proceed to Step 6 in "Data source configuration through a GUI." Optionally, you can further configure your driver by clicking on the following tabs. The following sections provide details for the fields specific to each configuration tab:

- General tab allows you to configure options that are required for creating a data source.
- Advanced tab allows you to configure advanced behavior.
- Schema Map tab allows you to configure mapping behavior.
- Security tab allows you to configure security settings.

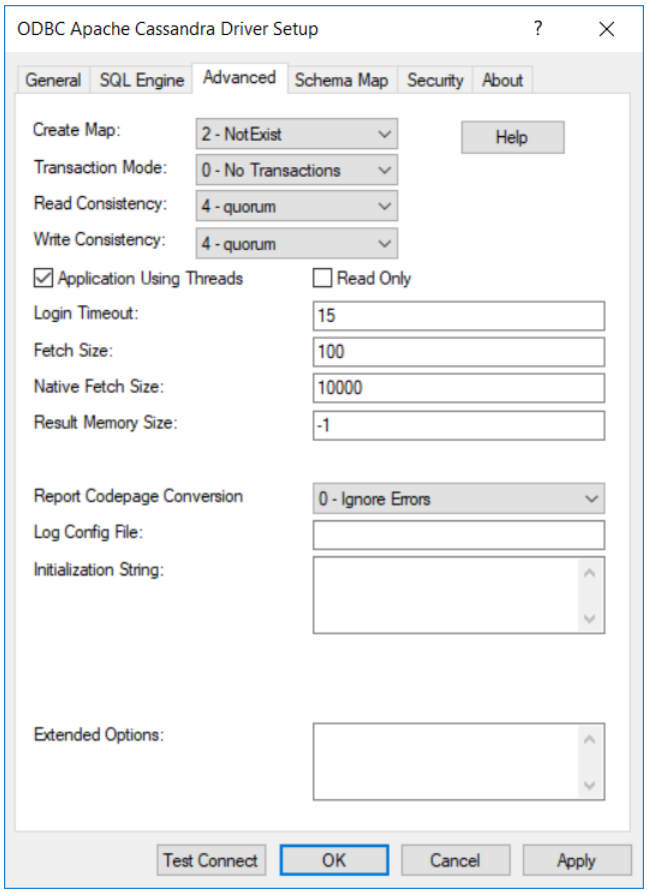
See also

Data source configuration through a GUI on page 54

Advanced tab

The Advanced Tab allows you to specify additional data source settings. The fields are optional unless otherwise noted. On this tab, provide values for the options in the following table; then, click **Apply**.

Figure 3: Advanced tab



Connection Options: Advanced	Description
Create Map on page 95	Determines whether the driver creates the internal files required for a relational view of the native data when establishing a connection. If set to **0 - No**, the driver uses the current group of internal files specified by the Schema Map connection option. If the files do not exist, the connection fails. If set to **1 - ForceNew**, the driver deletes the current group of files specified by the Schema Map connection option and creates new files in the same location. If set to **2 - NotExist**, the driver uses the current group of files specified by the Schema Map connection option. If the files do not exist, the driver creates them. **Default: 2 - NotExist**

Connection Options: Advanced	Description
Transaction Mode on page 115	Specifies how the driver handles manual transactions. If set to **0 - No Transactions**, the data source and the driver do not support transactions. Metadata indicates that the driver does not support transactions. If set to **1 - Ignore**, the data source does not support transactions and the driver always operates in auto-commit mode. **Default: 0 - No Transactions**
Read Consistency on page 108	Specifies how many replicas must respond to a read request before returning data to the client application. If set to **1 - one**, data is returned from the closest replica. This setting provides the highest availability, but increases the likelihood of stale data being read. If set to **4 - quorum**, data is returned after a quorum of replicas has responded from any data center. If set to **5 - all**, data is returned to the application after all replicas have responded. This setting provides the highest consistency and lowest availability. **Default: 4 - quorum**
Write Consistency on page 117	Determines the number of replicas on which the write must succeed before returning an acknowledgment to the client application. If set to **1 - one**, a write must succeed on at least one replica node. If set to **4 - quorum**, a write must succeed on a quorum of replica nodes. If set to **5 - all**, a write must succeed on all replica nodes in the cluster for that partition key. This setting provides the highest consistency and lowest availability. **Default: 4 - quorum**
Application Using Threads on page 90	Determines whether the driver works with applications using multiple ODBC threads. If disabled, the driver does not work with multi-threaded applications. If using the driver with single-threaded applications, this value avoids additional processing required for ODBC thread-safety standards. If set to enabled, the driver works with single-threaded and multi-threaded applications. **Default:** Enabled
Read Only on page 109	Specifies whether the connection has read-only access to the data source. If enabled, the connection has read-only access. If disabled, the connection is opened for read/write access, and you can use all commands supported by the product. **Default:** Disabled

Connection Options: Advanced	Description
Login Timeout on page 105	The number of seconds the driver waits for a connection to be established before returning control to the application and generating a timeout error. **Default:** `15`
Fetch Size on page 98	Specifies the number of rows that the driver processes before returning data to the application when executing a Select. If set to `0`, the driver fetches and processes all of the rows of the result before returning control to the application. If set to *x*, the driver limits the number of rows that may be processed and returned to the application for a single fetch request. **Default:** `100`
Native Fetch Size on page 106	Specifies the number of rows of data the driver attempts to fetch from the native data source on each request submitted to the server. If set to `0`, the driver requests that the server return all rows for each request submitted to the server. Block fetching is not used. If set to *x*, the driver attempts to fetch up to a maximum of the specified number of rows on each request submitted to the server. **Default:** `10000`
Result Memory Size on page 110	Specifies the maximum size, in megabytes, of an intermediate result set that the driver holds in memory. If set to `-1`, the maximum size of an intermediate result set that the driver holds in memory is determined by a percentage of the max Java heap size. When this threshold is reached, the driver writes a portion of the result set to disk. If set to `0`, the driver holds intermediate results in memory regardless of size. Setting Result Memory Size to `0` can increase performance for any result set that can easily fit within the JVM's free heap space, but can decrease performance for any result set that can barely fit within the JVM's free heap space. If set to *x*, the driver holds intermediate results in memory that are no larger than the size specified. When this threshold is reached, the driver writes a portion of the result set to disk. **Default:** `-1`

Connection Options: Advanced	Description
Report Codepage Conversion Errors on page 110	Specifies how the driver handles code page conversion errors that occur when a character cannot be converted from one character set to another. If set to **0 - Ignore Errors**, the driver substitutes 0x1A for each character that cannot be converted and does not return a warning or error. If set to **1 - Return Error**, the driver returns an error instead of substituting 0x1A for unconverted characters. If set to **2 - Return Warning**, the driver substitutes 0x1A for each character that cannot be converted and returns a warning. **Default: 0 - Ignore Errors**
Log Config File on page 105	Specifies the filename of the configuration file used to initialize the driver logging mechanism. **Default:** `ddlogging.properties`
Initialization String on page 101	One or multiple SQL commands to be executed by the driver after it has established the connection to the database and has performed all initialization for the connection. If the execution of a SQL command fails, the connection attempt also fails and the driver returns an error indicating which SQL command or commands failed. **Default:** None

Extended Options: Type a semi-colon separated list of connection options and their values. Use this configuration option to set the value of undocumented connection options that are provided by Progress DataDirect Customer Support. You can include any valid connection option in the Extended Options string, for example:

```
KeyspaceName=mykeyspace;UndocumentedOption1=value [;UndocumentedOption2=value;]
```

If the Extended Options string contains option values that are also set in the setup dialog or data source, the values of the options specified in the Extended Options string take precedence. However, connection options that are specified on a connection string override any option value specified in the Extended Options string.

If you finished configuring your driver, proceed to Step 6 on page 57 in "Data source configuration through a GUI." Optionally, you can further configure your driver by clicking on the following tabs. The following sections provide details on the fields specific to each configuration tab:

- General tab allows you to configure options that are required for creating a data source.
- SQL Engine tab allows you to configure the SQL engine's behavior.
- Schema Map tab allows you to configure mapping behavior.
- Security tab allows you to configure security settings.

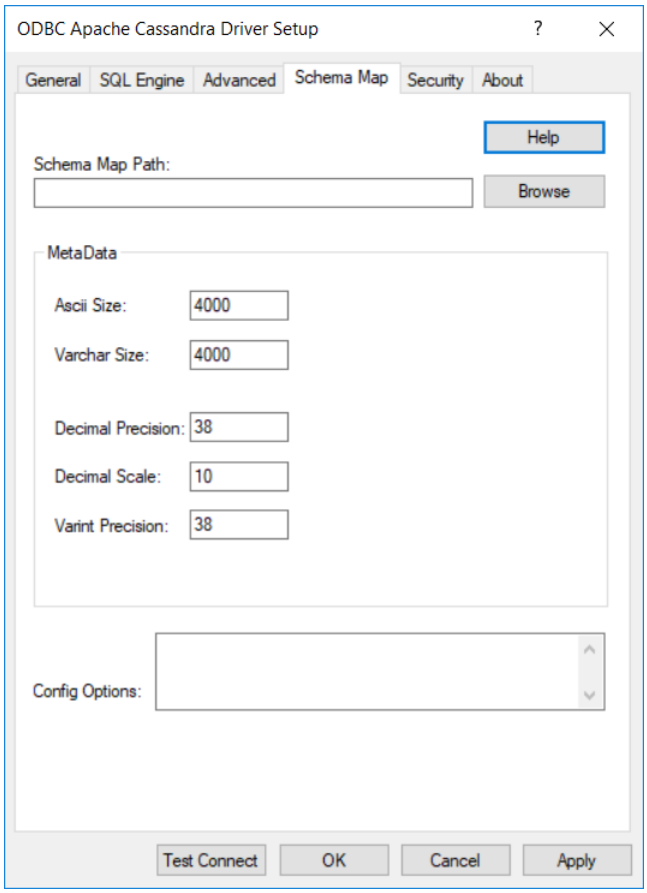
See also

Data source configuration through a GUI on page 54

Schema Map tab

The Schema Map tab allows you to configure the options that control the relational mapping of your data. The fields are optional unless otherwise noted. On this tab, provide values for the options in the following table; then, click **Apply**.

Figure 4: Schema Map tab



Connection Options: Advanced	Description
Schema Map on page 112	Specifies the name and location of the configuration file where the relational map of native data is written. The driver looks for this file when connecting to a server. If the file does not exist, the driver creates one. **Default:** *application_data_folder*\Local\Progress\DataDirect\Cassandra_Schema*host_name*.config
Ascii Size on page 91	Specifies the precision reported for ASCII columns in column and result-set metadata. **Default:** 4000

Connection Options: Advanced	Description
Varchar Size on page 116	Specifies the precision reported for Varchar columns in column and result-set metadata. **Default:** `4000`
Decimal Precision on page 96	Specifies the precision reported for Decimal columns in column and result-set metadata. **Default:** `38`
Decimal Scale on page 97	Specifies the maximum scale reported for Decimal columns in column and result-set metadata. **Default:** `10`
Varint Precision on page 116	Specifies the precision reported for Varint columns in column and result-set metadata. **Default:** `38`
Config Options on page 93	Determines how the mapping of the native data model to the relational data model is configured, customized, and updated. **Default:**None

If you finished configuring your driver, proceed to Step 6 in "Data source configuration through a GUI." Optionally, you can further configure your driver by clicking on the following tabs. The following sections provide details on the fields specific to each configuration tab:

- General tab allows you to configure options that are required for creating a data source.
- SQL Engine tab allows you to configure the SQL engine's behavior.
- Advanced tab allows you to configure advanced behavior.
- Security tab allows you to configure security settings.

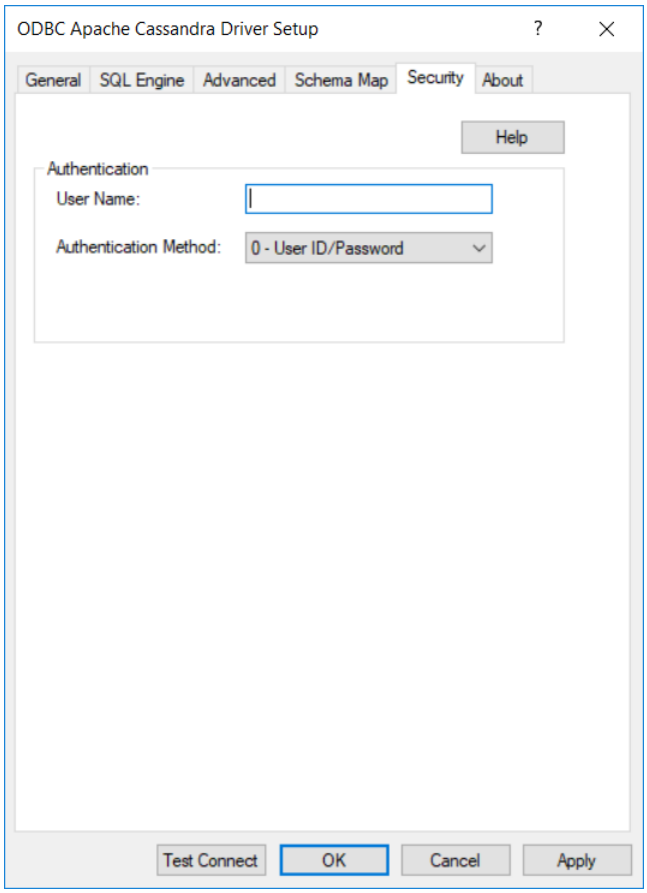
See also

Data source configuration through a GUI on page 54

Security tab

The Security tab allows you to specify your security settings. The fields are optional unless otherwise noted. On this tab, provide values for the options in the following table; then, click **Apply**.

Figure 5: Security tab



Connection Options: Advanced	Description
User Name on page 115	The default user ID that is used to connect to your database. **Default:** None
Authentication Method on page 92	Specifies the method the driver uses to authenticate the user to the server when a connection is established. If the specified authentication method is not supported by the database server, the connection fails and the driver generates an error. If set to **-1 - No Authentication**, the driver sends the user ID and password in clear text to the server for authentication. If set to **0 - User ID/Password**, the driver sends the user ID in clear text and an encrypted password to the server for authentication. **Default: 0 - User ID/Password**

If you finished configuring your driver, proceed to Step 6 in "Data source configuration through a GUI." Optionally, you can further configure your driver by clicking on the following tabs. The following sections provide details on the fields specific to each configuration tab:

- General tab allows you to configure options that are required for creating a data source.
- SQL Engine tab allows you to configure the SQL engine's behavior.
- Advanced tab allows you to configure advanced behavior.
- Schema Map tab allows you to configure mapping behavior.

See also

Data source configuration through a GUI on page 54

Using a connection string

If you want to use a connection string for connecting to a database, or if your application requires it, you must specify either a DSN (data source name), a File DSN, or a DSN-less connection in the string. The difference is whether you use the DSN=, FILEDSN=, or the DRIVER= keyword in the connection string, as described in the ODBC specification. A DSN or FILEDSN connection string tells the driver where to find the default connection information. Optionally, you may specify *attribute=value* pairs in the connection string to override the default values stored in the data source.

The DSN connection string has the form:

```
DSN=data_source_name[;attribute=value[;attribute=value]...]
```

The FILEDSN connection string has the form:

```
FILEDSN=filename.dsn[;attribute=value[;attribute=value]...]
```

The DSN-less connection string specifies a driver instead of a data source. All connection information must be entered in the connection string because the information is not stored in a data source.

The DSN-less connection string has the form:

```
DRIVER=[{]driver_name[}][;attribute=value[;attribute=value]...]
```

"Connection Option Descriptions" lists the long and short names for each attribute, as well as the initial default value when the driver is first installed. You can specify either long or short names in the connection string.

An example of a DSN connection string with overriding attribute values for Apache Cassandra for Linux/UNIX/Windows is:

```
DSN=Apache Cassandra;UID=JOHN;PWD=XYZZY
```

A FILEDSN connection string is similar except for the initial keyword:

```
FILEDSN=Apache Cassandra;UID=JOHN;PWD=XYZZY
```

A DSN-less connection string must provide all necessary connection information:

```
DRIVER=DataDirect 8.0 Apache Cassandra;UID=JOHN;PWD=XYZZY;HOST=CassandraServer;
PORT=9042;DB=Cassandra1;
SM=/home/users/jsmith/Progress/DataDirect/Cassandra_Schema/CassandraServer.config
```

See also

Connection option descriptions on page 87

Password Encryption Tool (UNIX/Linux only)

On UNIX and Linux, Progress DataDirect provides a Password Encryption Tool, called `ddencpwd`, that encrypts passwords for secure handling in connection strings and `odbc.ini` files. At connection, the driver decrypts these passwords and passes them to the data source as required. Passwords can be encrypted for any option, including:

- KeyPassword
- KeyStorePassword
- TrustStorePassword
- Password

To use the Password Encryption Tool:

1. From a command line, navigate to the directory containing the `ddencpwd` application. By default, this is *`install_directory`*`/tools`.
2. Enter the following command:

   `ddencpwd` *`password`*

   where:

   `password`

   is the password you want to encrypt.

3. The tool returns an encrypted password value. Specify the returned value for the corresponding attribute in the connection string or `odbc.ini` file. For example, if you encrypted the password for `KeyPassword`, specify the following in your connection string or datasource definition:

   `KeyPassword=`*`returned_value`*

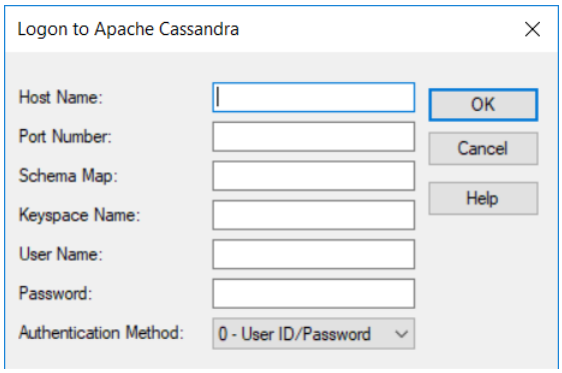
4. Repeat Steps 2 and 3 to encrypt additional passwords.
5. If using an `odbc.ini` file, save your file.

This completes this tutorial. You are now ready to connect using encrypted passwords.

Using a logon dialog box

Some ODBC applications display a logon dialog box when you are connecting to a data source. In these cases, the host name has already been specified.

Figure 6: Logon to Apache Cassandra dialog box



In this dialog box, provide the following information:

1. In the Host Name field, type the name or the IP address of the server to which you want to connect to which you want to connect.
2. In the Port Number field, type the port number of the server listener.
3. In the Schema Map field, type the name and location of the configuration file where the relational map of native data is written. See "Schema Map" for details.
4. In the Keyspace Name field, type the name of the keyspace to which you want connect.
5. Type your logon ID in the User Name field.
6. Type your password in the Password field.
7. From the Authentication Method drop-down box, select one the following:
   - For no authentication, select **-1 - No Authentication**.
   - To authenticate by passing the user ID in clear text and an encrypted password, select **0 - User ID/Password**.
8. Click **OK** to complete the logon.

See also

Schema Map on page 112

Performance considerations

The following connection options can enhance driver performance.

Application Using Threads (ApplicationUsingThreads): The driver coordinates single and concurrent database operations (operations from different threads) on a single connection by acquiring locks. Although locking prevents errors in the driver, it can decrease performance. If your single-threaded or multi-threaded applications are not sharing connections, the driver has no reason to coordinate operations. In this case, the ApplicationUsingThreads attribute should be disabled (set to `0`).

Note: If you are using a multi-threaded application, you must enable the Application Using Threads option.

Fetch Size (FetchSize) and **Native Fetch Size (NativeFetchSize)**: The connection options Fetch Size and Native Fetch Size can be used to adjust the trade-off between throughput and response time. In general, setting larger values for Fetch Size and Native Fetch Size will improve throughput, but can reduce response time.

For example, if an application attempts to fetch 100,000 rows from the native data source and Native Fetch Size is set to `500`, the driver must make 200 round trips across the network to get the 100,000 rows. If, however, Native Fetch Size is set to `10000`, the driver only needs to make 10 round trips to retrieve 100,000 rows. Network round trips are expensive, so generally, minimizing these round trips increases throughput.

For many applications, throughput is the primary performance measure, but for interactive applications, such as Web applications, response time (how fast the first set of data is returned) is more important than throughput. For example, suppose that you have a Web application that displays data 50 rows to a page and that, on average, you view three or four pages. Response time can be improved by setting Fetch Size to `50` (the number of rows displayed on a page) and Native Fetch Size to `200`. With these settings, the driver fetches all of the rows from the native data source that you would typically view in a single session and only processes the rows needed to display the first page.

Note: Fetch Size provides a suggestion to the driver as to the number of rows it should internally process before returning control to the application. The driver may fetch fewer rows to conserve memory when processing exceptionally wide rows.

JVM Arguments (JVMArgs): Used in conjunction with the Result Memory Size connection option, you can address memory and performance concerns by adjusting the max Java heap size using the JVM Arguments connection option. By increasing the max Java heap size, you increase the amount of data the driver accumulates in memory. This can reduce the likelihood of out-of-memory errors and improve performance by ensuring that result sets fit easily within the JVM's free heap space. In addition, when a limit is imposed by setting Result Memory Size to `-1`, increasing the max Java heap size can improve performance by reducing the need to write to disk, or removing it altogether.

Read Consistency (ReadConsistency): The Read Consistency connection option manages the trade-off between the consistency and availability of your data. By setting Read Consistency to `all`, data is returned to the application only after all replicas have responded. With this setting, the data returned is highly consistent. However, it may take longer for data to be returned to the application, and, in some scenarios, operation timeouts can occur. In contrast, setting Read Consistency to `quorum` (default) or `one` reduces the number of replicas required to respond to a read request. While the data may not be as consistent, results are returned more quickly to the application.

Result Memory Size (ResultMemorySize): Result Memory Size can affect performance in two main ways. First, if the size of the result set is larger than the value specified for Result Memory Size, the driver writes a portion of the result set to disk. Since writing to disk is an expensive operation, performance losses will be incurred. Second, when you remove any limit on the size of an intermediate result set by setting Result Memory Size to `0`, you can realize performance gains for result sets that easily fit within the JVM's free heap space. However, the same setting can diminish performance for result sets that barely fit within the JVM's free heap space.

Write Consistency (WriteConsistency): The Write Consistency connection option determines the number of replicas on which the write must succeed before returning an acknowledgment to the client application. By setting this option to `all`, a write must succeed on all replica nodes in the cluster for that partition key. This setting provides high consistency. However, it may take longer for the successful execution of a write operation. In contrast, setting Write Consistency to `quorum` (default) or `one` reduces the number of replicas that must acknowledge the completion of a write command. While the data may not be as consistent across clusters, the write operation is completed more quickly.

See also

Application Using Threads on page 90
Fetch Size on page 98
Native Fetch Size on page 106
JVM Arguments on page 102
Read Consistency on page 108
Result Memory Size on page 110
Write Consistency on page 117

Using the SQL engine server

Some applications may experience problems loading the JVM required for the SQL engine because the process exceeds the available heap space. If your application experiences problems loading the JVM, you can configure the driver to operate in server mode.

In direct mode, the driver operates with the SQL engine and JVM running in a single process. While in server mode, the driver's SQL engine runs in a separate process with its own JVM instead of trying to load the SQL engine and JVM in the same process used by the driver.

For Windows, the driver is configured to attempt to run in server mode first by default. However, if server mode is unavailable, the SQL engine will failover to run in direct mode. For non-Windows platforms, the driver operates in direct mode by default.

Note: You must be an administrator to start or stop the service, or to configure any settings for the service.

See the following sections for details on configuring the SQL engine server on your platform.

Configuring the SQL engine server on Windows

The following sections describe how to configure, start, and stop the SQL engine server on Windows platforms.

On Windows, the driver is configured to run in Auto mode by default. This means that driver attempts to run in server mode first; however, if server mode is unavailable, the SQL engine will failover to run in direct mode.

Configuring server mode on Windows

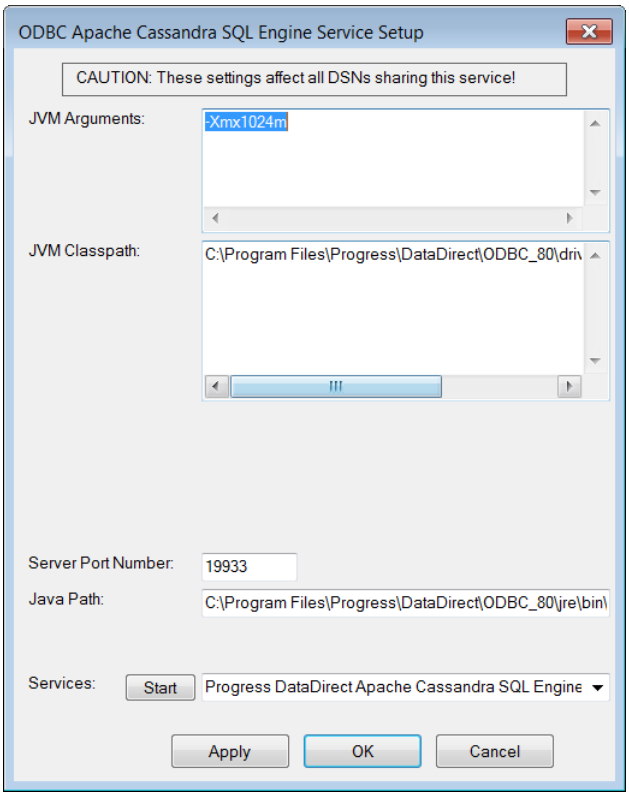
1. Set the SQL Engine Mode connection option to a value of **0 - Auto** or **1 - Server**. All fields on the SQL Engine tab become read only, and the **Edit Server Settings** button appears.

   Note: Server mode is enabled when the SQL Engine Mode connection option is set to **0 - Auto** or **1- Server**. When set **0 - Auto**, the SQL engine attempts to run in server mode first, but will failover to direct mode if server mode is unavailable. When set to **1 - Server**, the SQL engine mode runs exclusively in server mode.

   ---

2. Click **Edit Server Setting** to display the ODBC Cassandra SQL Engine Service Setup dialog box. Use this dialog box to define settings for Server Mode and to start and stop the SQL engine service.

   The SQL Engine Service Setup dialog box appears.



JVM Arguments: A string that contains the arguments that are passed to the JVM that the driver is starting. The location of the JVM must be specified on your PATH. See "JVM Arguments."

JVM Class Path: Specifies the CLASSPATH for the JVM used by the driver. See "JVM Classpath."

Server Port Number: Specifies a valid port on which the SQL engine listens for requests from the driver. By default, the server listens on port 19933 for 64-bit installations and 19934 for 32-bit installations. See "Server Port Number" for more information.

Java Path: Specifies fully qualified path to the JVM executable that you want to use to run the SQL engine server. The path must not contain double quotation marks.

Services: Shows the Cassandra ODBC SQL engine service that runs as a separate process instead of being loaded within the process of an ODBC application.

Start (**Stop**): Starts or stops the Cassandra service. A message window is displayed, confirming that the Cassandra service was started or stopped.

Apply: Applies the changes.

Note: After the initial configuration, in order for changes to these connection option values to take effect, you must restart the SQL engine server.

---

3. When you complete your changes, click **Apply**.
4. Click **OK** to save the changes and return to the SQL Engine tab or click **Cancel**.

See also

JVM Arguments on page 102
JVM Classpath on page 103
Server Port Number on page 113

Starting the SQL engine server on Windows

In server mode, you must start the SQL engine server before using the driver. Before starting the SQL engine server, choose a directory to store the local database files. Make sure that you have the correct permissions to write to this directory.

By default, the JVM Classpath is set to the `cassandra.jar` file in the installation directory.

To start the SQL engine server:

1. Start the ODBC Administrator by selecting its icon from the Progress DataDirect for ODBC program group.
2. Select a tab:

   - **User DSN**: If you are configuring an existing user data source, select the data source name and click **Configure** to display the driver Setup dialog box.

   If you are configuring a new user data source, click **Add** to display a list of installed drivers. Select the driver and click **Finish** to display the driver Setup dialog box.

   - **System DSN**: If you are configuring an existing system data source, select the data source name and click **Configure** to display the driver Setup dialog box.

   If you are configuring a new system data source, click **Add** to display a list of installed drivers. Select the driver and click **Finish** to display the driver Setup dialog box.

   - **File DSN**: If you are configuring an existing file data source, select the data source file and click **Configure** to display the driver Setup dialog box.

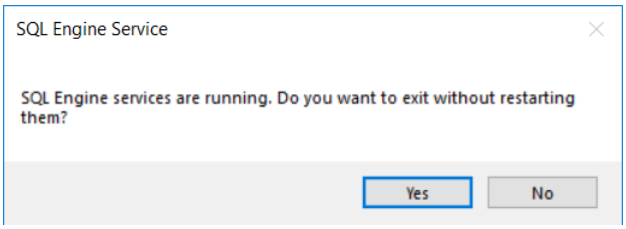
   If you are configuring a new file data source, click **Add** to display a list of installed drivers; then, select a driver. Click **Advanced** if you want to specify attributes; otherwise, click **Next** to proceed. Specify a name for the data source and click **Next**. Verify the data source information; then, click **Finish** to display the driver Setup dialog box.

3. On the ODBC Cassandra Driver Setup dialog box, select the **SQL Engine** tab; then, select **0 - Auto** or **1- Server** from the SQL Engine Mode drop-down list.

   Note: Server mode is enabled when the SQL Engine Mode connection option is set to **0 - Auto** or **1- Server**. When set **0 - Auto**, the SQL engine attempts to run in server mode first, but will failover to direct mode if server mode is unavailable. When set to **1 - Server**, the SQL engine mode runs exclusively in server mode.

4. Click **Edit Server Settings**.
5. When you complete your changes, click **Apply**.
6. Verify that Progress DataDirect Cassandra SQL Engine is selected in the Services drop-down list, and then, click **Start** to start the service. A message window appears to confirm that the service is running. Click **OK**.
7. Click **OK** to close the ODBC Cassandra SQL Engine Service Setup dialog box.

   Note: If you made changes after starting the service, a message window is displayed:



If you want the service to run with the new settings, click **No**. Then, click **Stop** to stop the service, and then click **Start** to restart the service. Then, click **OK** to close the ODBC Cassandra SQL Engine Service Setup dialog box.

Stopping the SQL engine server on Windows

To stop the SQL engine server:

1. Open the ODBC Cassandra Driver Setup dialog box and select the SQL Engine tab.
2. Select **0 - Auto** or **1 - Server** from the SQL Engine Mode drop-down list. Then, click **Edit Server Settings**.

Note: Server mode is enabled when the SQL Engine Mode connection option is set to **0 - Auto** or **1- Server**. When set **0 - Auto**, the SQL engine attempts to run in server mode first, but will failover to direct mode if server mode is unavailable. When set to **1 - Server**, the SQL engine mode runs exclusively in server mode.

---

3. Click **Stop** to stop the service. A message window appears to confirm that the service is stopped. Click **OK**.
4. Click **OK** to close the ODBC Cassandra SQL Engine Service Setup dialog box.

Configuring the SQL engine server on UNIX/Linux

The following sections describe how to configure, start, and stop the SQL engine server on UNIX and Linux platforms.

By default, the driver operates in direct mode by default on UNIX and Linux platforms.

Configuring and starting the SQL engine server on UNIX/Linux

In server mode, you must start the SQL engine server before using the driver. Be aware that you must have permissions to write to the directory specified by the SchemaMap option to start the SQL engine server.

To configure the SQL engine server, specify values for the Java options in the following JVM argument:

```
java -Xmx<heap_size>m -cp "<jvm_classpath>" com.ddtek.cassandracloud.sql.Server
-port <port_number> -Dhttp.proxyHost=<proxy_host> -Dhttp.proxPort=<proxy_port>
-Dhttp.proxyUser=<proxy_user> -Dhttp.proxyPassword=<proxy_password>
```

See the "SQL engine server Java options" table for a description of these options.

For example:

```
java -Xmx1024m -cp "/opt/Progress/DataDirect/ODBC_80_64bit/java/lib/cassandra.jar"
com.ddtek.cassandracloud.sql.Server -port 19933 -Dhttp.proxyHost=myhost@mydomain.com
-Dhttp.proxPort=12345 -Dhttp.proxyUser=JohnQPublic -Dhttp.proxyPassword=secret
```

To start the SQL engine service, execute the JVM Argument after configuring the Java options. A confirmation message is returned once the server is online.

Note: After the initial configuration, in order for changes to these connection option values to take effect, you must restart the SQL engine server.

Table 14: SQL engine server Java options

Java Option	Description
Required Java Options	
`-cp`	Specifies the CLASSPATH for the Java Virtual Machine (JVM) used by the driver. The CLASSPATH is the search string the JVM uses to locate the Java jar files the driver needs. The Cassandra driver's JVM is located on the following path: *`install_dir`*`/java/lib/cassandra.jar`
`-port`	Specifies a valid port on which the SQL engine listens for requests from the driver. We recommend specifying one of the following values: • `19934` (32-bit drivers) • `19933` (64-bit drivers)
Optional Java Options	
`-Xmx`	Specifies the maximum memory heap size, in megabytes, for the JVM. The default size is determined by your JVM. We recommend specifying a size no smaller than `1024`. **Note:** Although this option is not required to start the SQL engine server, we highly recommend specifying a value.
`-Dhttp.proxyHost`	Specifies the Hostname of the Proxy Server. The value specified can be a host name, a fully qualified domain name, or an IPv4 or IPv6 address.
`-Dhttp.proxyPort`	Specifies the port number where the Proxy Server is listening for HTTP and/or HTTPS requests.
`-Dhttp.proxyUser`	Specifies the user name needed to connect to the Proxy Server.
`-Dhttp.proxyPassword`	Specifies the password needed to connect to the Proxy Server.

Stopping the SQL engine server on UNIX/Linux

To stop the SQL engine server, choose one of the following:

- Using an application, execute SHUTDOWN SQL.
- From a command line, press `Ctrl` + `C`.

If successful, a message is returned to confirm that the service has stopped.

Configuring Java logging for the SQL engine server

Java logging can be configured by placing a logging configuration file named `ddlog.properties` in the directory specified by the Schema Map option. The simple way to create one of these is to make a copy of the `ddlog.properties` file, which is located in your driver installation directory, in the *`install_dir`*`/Sample/Example` subdirectory.

For more information, refer to "Logging for Java components" in the *Progress DataDirect for ODBC Drivers Reference*.

See also

Schema Map on page 112

Using failover in a cluster

To ensure availability to Cassandra data sources, the driver supports connection failover and client load balancing when connecting to clusters:

- *Connection failover* provides protection for new connections to a cluster. If a member node of the cluster is unavailable, for example, because of hardware failure or traffic overload, the driver fails over to another member node to attempt the connection. If a connection to a node is dropped, the driver does not fail over the connection.
- *Client load balancing* helps distribute new connections in your environment so that no one server is overwhelmed with connection requests. When client load balancing is enabled, the order in which member nodes are tried is random.

You can enable connection failover and load balancing by specifying a list of the member nodes for your cluster using the Cluster Nodes (ClusterNodes) option. The list should be comprised of a comma-separated server names or IP addresses for your member nodes and their corresponding port numbers. Note that IPv6 values must be wrapped in double quotation marks. For example:

```
ClusterNodes=server1:9042,255.125.1.11:9043,"2001:DB8:0000:0000:8:800:200C:417A":9044
```

Note: Specifying a value for the Cluster Nodes option can be used in place of or in addition to the values of the Host Name (HostName) and Port Number (PortNumber) options. If values are specified all of these options, the node specified by the Host Name and Port Number are included in the list of nodes to which the driver randomly attempts to connect.

When connection failover and load balancing are enabled, the driver randomly selects a node from the list to first attempt a connection. If that connection fails, the driver again randomly selects another node from this list until all the nodes in the list have been tried or a connection is successfully established.

See also

Cluster Nodes on page 92

Using identifiers

Identifiers are used to refer to objects exposed by the driver, such as tables and columns. The driver supports both quoted and unquoted identifiers for naming objects. The maximum length of both quoted and unquoted identifiers is 48 characters for table names and 128 characters for column names. Quoted identifiers must be enclosed in double quotation marks (""). The characters supported in quoted identifiers depends on the version of Cassandra being used. For details on valid characters, refer to the Cassandra documentation for your database version.

Naming conflicts can arise from restrictions imposed by third party applications, from the normalization of native data, or from the truncation of object names. The driver avoids naming conflicts by appending an underscore separator and integer (for example, `_1`) to identifiers with the same name. For example, if a third party application restricts the naming of three columns such that each column retains the name `address`, the driver would expose the columns in the following manner:

- `ADDRESS`
- `ADDRESS_1`
- `ADDRESS_2`

Isolation and lock levels supported

The driver supports isolation level 0 (Read Uncommitted).

Refer to "Locking and isolation levels" in the *Progress DataDirect for ODBC Drivers Reference* for details.

See also

Transaction Mode on page 115

Number of connections and statements supported

The Apache Cassandra driver supports multiple connections and multiple statements per connection.

Unicode support

The driver is fully Unicode enabled. On UNIX and Linux platforms, the driver supports both UTF-8 and UTF-16. On Windows platforms, the driver supports UCS-2/UTF-16 only.

The driver supports the Unicode ODBC W (Wide) function calls, such as SQLConnectW. This allows the Driver Manager to transmit these calls directly to the driver. Otherwise, the Driver Manager would incur the additional overhead of converting the W calls to ANSI function calls, and vice versa.

See "UTF-16 applications on UNIX and Linux" for related details.

Also, refer to "Internationalization, localization, and Unicode" in the *Progress DataDirect for ODBC Drivers Reference* for a more detailed explanation of Unicode.

See also

UTF-16 applications on UNIX and Linux on page 54

Binding parameter markers

An ODBC application can prepare a query that contains dynamic parameters. Each parameter in a SQL statement must be associated, or bound, to a variable in the application before the statement is executed. When the application binds a variable to a parameter, it describes that variable and that parameter to the driver. Therefore, the application must supply the following information:

- The data type of the variable that the application maps to the dynamic parameter
- The SQL data type of the dynamic parameter (the data type that the database system assigned to the parameter marker)

The two data types are identified separately using the SQLBindParameter function. You can also use descriptor APIs as described in the Descriptor section of the ODBC specification (version 3.0 and higher).

The driver relies on the binding of parameters to know how to send information to the database system in its native format. If an application furnishes incorrect parameter binding information to the ODBC driver, the results will be unpredictable. For example, the statement might not be executed correctly.

To ensure interoperability, your driver uses only the parameter binding information that is provided by the application.

Parameter metadata support

The driver supports returning parameter metadata as described in this section.

Insert and Update statements

The driver supports returning parameter metadata for the following forms of Insert and Update statements:

- `INSERT INTO FOO VALUES(?, ?, ?)`
- `INSERT INTO FOO (COL1, COL2, COL3) VALUES(?, ?, ?)`
- `UPDATE FOO SET COL1=?, COL2=?, COL3=? WHERE COL1 operator ? [{AND | OR} COL2 operator ?]`

where:

operator

is any of the following SQL operators:

=, <, >, <=, >=, and <>

.

Select statements

The driver supports returning parameter metadata for Select statements that contain parameters in ANSI SQL 92 entry-level predicates, for example, such as COMPARISON, BETWEEN, IN, LIKE, and EXISTS predicate constructs. Refer to the ANSI SQL reference for detailed syntax.

Parameter metadata can be returned for a Select statement if one of the following conditions is true:

- The statement contains a predicate value expression that can be targeted against the source tables in the associated FROM clause. For example:

  ```
  SELECT * FROM FOO WHERE BAR > ?
  ```

  In this case, the value expression "BAR" can be targeted against the table "FOO" to determine the appropriate metadata for the parameter.

- The statement contains a predicate value expression part that is a nested query. The nested query's metadata must describe a single column. For example:

  ```
  SELECT * FROM FOO WHERE (SELECT X FROM Y    WHERE Z = 1) < ?
  ```

The following Select statements show further examples for which parameter metadata can be returned:

```
SELECT COL1, COL2 FROM FOO WHERE COL1 = ? AND COL2 > ?
SELECT ... WHERE COLNAME = (SELECT COL2 FROM T2 WHERE COL3 = ?)
SELECT ... WHERE COLNAME LIKE ?
SELECT ... WHERE COLNAME BETWEEN ? AND ?
SELECT ... WHERE COLNAME IN (?, ?, ?)
SELECT ... WHERE EXISTS(SELECT ... FROM T2 WHERE COL1 < ?)
```

ANSI SQL 92 entry-level predicates in a WHERE clause containing GROUP BY, HAVING, or ORDER BY statements are supported. For example:

```
SELECT * FROM T1 WHERE COL = ? ORDER BY 1
```

Joins are supported. For example:

```
SELECT * FROM T1,T2 WHERE T1.COL1 = ?
```

Fully qualified names and aliases are supported. For example:

```
SELECT A, B, C, D FROM T1 AS A, T2 AS B WHERE A.A = ? AND B.B = ?
```

Operation timeouts

Cassandra imposes timeouts on read and write operations to prevent a given operation from negatively impacting the performance of the cluster. If you encounter an operation timeout, you can take the following actions to promote operation success.

- Adjust the Read Consistency (ReadConsistency) connection option. You can speed up a query by reducing the number of replicas required to respond to a read request. Therefore, you can reduce the likelihood of a timeout by setting Read Consistency to a value that requires fewer replicas to respond.
- Adjust the Write Consistency (WriteConsistency) connection option. You can speed up a write operation by reducing the number of replicas required to acknowledge success. Therefore, you can reduce the

likelihood of a timeout by setting Write Consistency to a value that requires fewer replicas to acknowledge the execution of the write operation.

- Decrease the value of the Native Fetch Size (NativeFetchSize) connection option. By decreasing Native Fetch Size, you reduce the amount of data that must be transmitted between the driver and the native data source. For read operations, the smaller the chunks of data requested, the faster the cluster can assemble results for transmission to the driver. For write operations, smaller chunks of data allow the driver to communicate more efficiently with the native data source and thus expedite write operations.

  Note: Setting Native Fetch Size too low negatively impacts performance by requiring unnecessary round trips across the network.

- Optimize your query by taking one or more of the following actions.

  1. Limit the number of results returned.
  2. Add indexes to Cassandra tables and base operations on indexes as appropriate.
  3. Use Where clause filtering that can be pushed down to Cassandra, allowing operations to be evaluated and handled quickly. Refer to the following DataStax Web pages for more information about Where clause functionality and limitations.

     - A deep look at the CQL WHERE clause
     - Filtering data using WHERE

- Adjust Cassandra network timeout settings in the `cassandra.yaml` configuration file. These settings can be adjusted to promote read operation success by increasing the size of the timeout window. Refer to your Apache Cassandra documentation for details.

See also

Read Consistency on page 108
Write Consistency on page 117
Native Fetch Size on page 106
Where clause on page 129

Out-of-memory issues

When processing large sets of data, out-of-memory errors can occur when the size of an intermediate result exceeds the available memory allocated to the JVM. If you are encountering these errors, you can tune Fetch Size, Result Memory Size, and JVM Arguments connection options to fit your environment:

- Reduce Fetch Size to reduce demands on the driver's internal memory. By lowering the maximum number of rows as specified by Fetch Size, you lower the number of rows the driver is required to process before returning data to the application. Thus, you reduce demands on the driver's internal memory, and, in turn, decrease the likelihood of out-of-memory errors.
- To tune Result Memory Size, decrease the value specified until results are successfully returned. Intermediate results larger than the specified setting will be written to disk as opposed to held in memory. When configured correctly, this avoids memory limitations by not relying on memory to process larger intermediate results. Be aware that while writing to disk reduces the risk of out-of-memory errors, it also negatively impacts performance. For optimal performance, decrease this value only to a size necessary to avoid errors.

Note: By default, Result Memory Size is set to `-1`, which sets the maximum size of intermediate results held in memory to a percentage of the max Java heap size. If you received errors using the default configuration, use the max Java heap size divided by 4 as a starting point when tuning this option.

- Increase the JVM heap size using the JVM Arguments connection option. By increasing the max Java heap size, you increase the amount of data the driver can accumulate in memory and avoid out-of-memory errors.

See "Fetch Size", "Result Memory Size", and "JVM Arguments" for additional information.

See also

Fetch Size on page 98
Result Memory Size on page 110
JVM Arguments on page 102

Packet logging

The driver code includes a packet logging mechanism that allows you to log TCP packets transmitted between your driver and database over the network layer. The logs compiled from can then be analyzed and used to troubleshoot issues. You can enable and configure logging using driver connection options.

Note: The packet logging mechanism is supported only for drivers that transmit TCP packets. Refer to "Packet Logging" in the *Progress DataDirect for ODBC Drivers Reference* for a list of supported drivers.

See the following "Packet Logging Connection options" section for a list of connection options used to configure packet logging.

To enable TCP packet logging:

1. Configure and enable packet logging using one of the following methods:
   - Driver setup dialog (Windows)
   - odbc.ini file (UNIX/Linux)
   - Connection string

   See the following "Configuring and enabling packet logging" section for details.
2. Start your application and reproduce the issue.
3. Stop the application and disable packet logging.
4. Send your logs to Technical Support for analysis. Optionally, you can view your logs using a text editor.

Configuring and enabling packet logging

The following driver configuration methods can be used to enable and configure packet logging. Note that only the `EnablePacketLogging` connection option is required to enable packet logging. If you do not specify values for the other connection options for packet logging, the default behavior is used.

Driver setup dialog (Windows)

You can specify connection options for packet logging in the Extended Options field of the **Advanced** tab. For example:

```
EnablePacketLogging=1;PacketLoggingFilePrefix=C:\temp\myPacketLog;
PacketLoggingMaxFileSize=7500
```

odbc.ini file (UNIX/Linux)

In your data source definition in the `[ODBC Data Sources]` section of the system information file, you can specify connection options that control packet logging.

```
[Apache Cassandra]
Driver=ODBCHOME/lib/ivcsndr28.so
Description=DataDirect 8.0 Apache Cassandra
...
EnablePacketLogging=1
...
HostName=CassandraServer
...
LogonID=JOHN
...
KeyspaceName=MyKeyspace
...
PacketLoggingFilePrefix=/tmp/myPacketLog
...
PacketLoggingMaxFileSize=102400
...
PacketLoggingMaxNumFiles=10
...
Password=secret
...
PortNumber=9042
...
SchemaMap=/home/users/jsmith/Progress/DataDirect/Cassandra_Schema/CassandraServer.config
...
```

Connection string

You can specify connection options that configure packet logging in connection strings.

```
DRIVER={DataDirect 8.0 Apache Cassandra};HostName=CassandraServer;PortNumber=9042;
KeyspaceName=MyKeyspace;LogonID=JOHN;Password=secret;EnablePacketLogging=1;
PacketLoggingFilePrefix=/tmp/myPacketLog;
SchemaMap=/home/users/jsmith/Progress/DataDirect/Cassandra_Schema/CassandraServer.config
```

Packet logging connection options

The following table describes the connection options used to configure packet logging.

Table 15: Packet Logging Connection Options

Option	Description
`EnablePacketLogging`	If set to `0`, packet logging is disabled. This is the default. If set to `1`, packet logging is enabled. If set to `2`, packet logging is enabled, but the generated log file does not contain packet data. This value is typically used for performance testing. (Windows only) If set to `5`, packet logging and ODBC tracing are enabled. If set to `6`, packet logging and ODBC tracing are enabled, but the log file for packet logging does not contain data.
`PacketLoggingFlush`	If set to `0`, the operating system determines when to write the log content stored in memory to disk. This is the default. If set `1`, the driver determines when to write the log content stored in memory to disk. If set to `2`, the content of memory is written to a the log file after each write. This setting provides a more complete logging history in the event of a crash, but can incur a performance penalty.
`PacketLoggingFilePrefix`	Specifies the path and prefix name of the log file. If no path is specified, the trace log resides in the working directory of the application you are using. For example: • `/tmp/myLogFile` (UNIX/Linux) • `C:\temp\myLogFile` (Windows) The above examples would generate a file named `myLogFile`*`YYYYMMDDhhmmssxxx_nn`*`.out` in the `temp` directory. If you do not specify a value for this option, the driver creates log files in the working directory using the following form: `pkt`*`YYYYMMDDhhmmssxxx_nn`*`.out`.
`PacketLoggingMaxFileSize`	Specifies the file size limit (in KB) of the log file. Once this file size limit is reached, a new log file is created and logging continues. The default is `102400`. Note that subsequent files are named by appending sequential numbers, starting at `1`, to the end of the original file name, for example, `myLog<`*`timestamp`*`>_1.out`, `myLog<`*`timestamp`*`>_2.out`, and so on.

Option	Description
PacketLoggingMaxNumFiles	Specifies the maximum number of log files that can be created. The default is 10. Once the maximum number of log files is created, the logging mechanism reopens the first file in the sequence, deletes the content, and continues logging in that file until the file size limit is reached, after which it repeats the process with the next file in the sequence.
PacketLoggingMemBuffSize	Specifies the maximum amount of memory, in kiloybtes, to use when writing packet logging. The default is 1024.

5

Connection option descriptions

The following connection option descriptions are listed alphabetically by the GUI name that appears on the driver Setup dialog box. The connection string attribute name, along with its short name, is listed immediately underneath the GUI name. For example:

Application Using Threads

Attribute

ApplicationUsingThreads (AUT)

In most cases, the GUI name and the attribute name are the same; however, some exceptions exist. If you need to look up an option by its connection string attribute name, please refer to the alphabetical table of connection string attribute names.

Also, a few connection string attributes, for example, Password, do not have equivalent options that appear on the GUI. They are in the list of descriptions alphabetically by their attribute names.

Note: The driver does not support specifying values for the same connection option multiple times in a connection string or DSN. If a value is specified using the same attribute multiple times or using both long and short attributes, the connection may fail or the driver may not behave as intended.

The following table lists the connection string attributes supported by the driver for Apache Cassandra.

Table 16: Apache Cassandra Attribute Names

Attribute (Short Name)	Default
ApplicationUsingThreads (AUT)	1 (Enabled)
AsciiSize (ASZ)	4000

Attribute (Short Name)	Default
AuthenticationMethod (AM)	`0` (User Id/Password)
ClusterNodes (CN)	None
ConfigOptions (CO)	None
CreateMap (CM)	`2` (NotExist)
DataSourceName (DSN)	None
DecimalPrecision (DP)	`38`
DecimalScale (DS)	`10`
Description (n/a)	None
FetchSize (FS)	`100` (rows)
HostName (HOST)	None
IANAAppCodePage (IACP) (UNIX and Linux only)	`4` (ISO 8559-1 Latin-1)
InitializationString (IS)	None

JVMArgs (JVMA)	For the 32-bit driver:
	`-Xmx256m`
	For the 64-bit driver:
	`-Xmx1024m`

JVMClasspath (JVMC)	The default is an empty string, which means that the driver automatically detects the CLASSPATHs for all ODBC drivers installed on your machine and specifies them when launching the JVM.
KeyspaceName (KSN)	`system`
LogConfigFile (LCF)	None
LoginTimeout (LT)	`15`
LogonID (UID)	None
NativeFetchSize (NFS)	`10000` (rows)
Password (PWD)	None
PortNumber (PORT)	`9042`
ReadConsistency (RC)	`4` (quorum)

Attribute (Short Name)	Default
ReadOnly (RO)	0 (Disabled)
ReportCodepageConversionErrors (RCCE)	0 (Ignore Errors)
ResultMemorySize (RMS)	-1
SchemaMap (SMP)	Default value depends on environment
ServerPortNumber (SPN)	For the 32-bit driver: 19934 For the 64-bit driver: 19933
SQLEngineMode (SEM)	For Windows: 0 (Auto) For UNIX/Linux: 2 (Direct)
TransactionMode (TM)	0 (No Transactions)
VarcharSize (VCS)	4000
VarintPrecision (VP)	38
WriteConsistency (WC)	4 (quorum)

For details, see the following topics:

- Application Using Threads
- Ascii Size
- Authentication Method
- Cluster Nodes
- Config Options
- Create Map
- Data Source Name
- Decimal Precision
- Decimal Scale
- Description
- Fetch Size
- Host Name

Application Using Threads

Attribute

ApplicationUsingThreads (AUT)

Purpose

Determines whether the driver uses internal locking mechanisms to ensure that multiple application threads work appropriately when they share an ODBC connection.

Valid Values

`0 | 1`

Behavior

If set to `1` (Enabled), the driver works with single-threaded and multi-threaded applications. This setting ensures that only a single thread is using an ODBC connection at a time. Until the connection becomes available, attempts by other threads to use the same connection are pended. Use this setting when your application includes multiple threads which share an ODBC connection and the application itself does not synchronize access to the connection.

If set to `0` (Disabled), the driver takes no action to prevent multiple application threads from using the same connection concurrently. This setting avoids the additional processing required for ODBC thread-safety standards. Use this setting when:

- The application is single threaded
- The application is multi-threaded and the threads each use their own connection to the database
- The application is multi-threaded and the threads are not sending statement-executing requests for multiple statements to a single connection at the same time

Notes

- Depending on the behavior of your application, this connection option can affect performance.

Default

`1` (Enabled)

GUI Tab

Advanced tab

See also

Performance considerations on page 70

Ascii Size

Attribute

AsciiSize (ASZ)

Purpose

Specifies the precision reported for ASCII columns in column and result-set metadata. This option allows you to set the precision for ASCII columns when using an application that does not support unbounded data types.

Valid Values

x

where:

x

is an integer greater than 0 (zero).

Default
4000

Notes

- In most scenarios, an error is returned if the size of an ASCII value specified in a statement exceeds the precision determined by this option. However, when executing a Select statement, the driver will return data containing values that are larger than the specified precision.

GUI Tab

Schema Map tab

Authentication Method

Attribute

AuthenticationMethod (AM)

Purpose

Specifies the method the driver uses to authenticate the user to the server when a connection is established. If the specified authentication method is not supported by the database server, the connection fails and the driver generates an error.

Valid Values

`-1` | `0`

Behavior

If set to `-1` (No Authentication), the driver does not attempt to authenticate with the server.

If set to `0` (User ID/Password), the driver sends the user ID in clear text and an encrypted password to the server for authentication.

Default

`0` (User ID/Password)

GUI Tab

Security tab

Cluster Nodes

Attribute

ClusterNodes (CN)

Purpose

A comma-separated list of member nodes in your cluster to which the driver attempts to connect. Specifying a value for this option enables connect time failover and load balancing. For details, see "Using failover in a cluster."

Valid Values

host_name | *IP_address*:*port_number* [, ...]

where:

host_name

is the name of a server for a member node in the replica-set cluster to which you want to connect.

IP_address

is the IP address of the server for a member node in the replica-set cluster to which you want to connect. The IP address can be specified in IPv4 or IPv6 format. Note that IPv6 values must be wrapped in double-quotation marks.

port_number

is the port number of the server listener for the corresponding member node.

For example:

```
server1:9042,255.125.1.11:9043,"2001:DB8:0000:0000:8:800:200C:417A":9044
```

Notes

- When specifying a value for this option, the driver randomly selects from the list for a server node to first attempt a connection. If that connection fails, the driver again randomly selects another server node from this list until all servers in the list have been tried or a connection is successfully established.

Default

None

GUI Tab

General tab

See Also

Using failover in a cluster on page 77

Config Options

Attribute

ConfigOptions (CO)

Purpose

Determines how the mapping of the native data model to the relational data model is configured, customized, and updated.

This option is primarily used for initial configuration of the driver for a particular user. It is not intended for use with every connection. By default, the driver configures itself and this option is normally not needed. If Config Options is specified on a connection after the initial configuration, the values specified for Config Options must match the values specified for the initial configuration.

Valid Values

`{ key = value [;  key = value ]}`

where:

key

is one of the following config options:

- `SchemaFormat`

value

specifies the setting for this configuration option.

Example

The value must be enclosed in curly brackets. For example:

```
{SchemaFormat=NormalizeNonFrozen}
```

Default

None

GUI Tab

Schema Map tab

SchemaFormat (config option)

Attribute

SchemaFormat

Purpose

Determines how the driver maps collections to the relational view of your data.

Valid Values

`NormalizeAll` | `NormalizeNonFrozen` | `NormalizeNone`

Behavior

If set to `NormalizeAll`, the driver normalizes all collection types when generating a relational view of your data. Collections are mapped to as child tables with foreign key relationships to parent tables. See "Complex Type Normalization" for more details.

If set to `NormalizeNonFrozen`, the driver normalizes collections not labeled FROZEN when generating a relational view of your data. Collection fields that are labeled FROZEN are returned as JSON strings in the table in which they are defined.

If set to `NormalizeNone`, the driver does not normalize any collection types when generating a relational view of your data. All collection fields are returned as JSON formatted strings in the table in which they are defined.

Default

`NormalizeAll`

GUI Tab

The values for config options are specified in the Config Options field on the Schema Map tab.

Create Map

Attribute

CreateMap (CM)

Purpose

Determines whether the driver creates the internal files required for a relational view of the native data when establishing a connection.

Valid Values

`0 | 1 | 2`

Behavior

If set to `0` (No), the driver uses the current group of internal files specified by the Schema Map connection option. If the files do not exist, the connection fails.

If set to `1` (ForceNew), the driver deletes the current group of files specified by the Schema Map connection option and creates new files in the same location.

If set to `2` (NotExist), the driver uses the current group of files specified by the Schema Map connection option. If the files do not exist, the driver creates them.

Notes

- The internal files share the same directory as the schema map's configuration file. This directory is specified by the value you enter for the Schema Map connection option.

Default

2 (NotExist)

GUI Tab

Advanced tab

Data Source Name

Attribute

DataSourceName (DSN)

Purpose

Specifies the name of a data source in your Windows Registry or `odbc.ini` file.

Valid Values

`string`

where:

`string`

is the name of a data source.

Default

None

GUI Tab

General tab

Decimal Precision

Attribute

DecimalPrecision (DP)

Purpose

Specifies the precision reported for Decimal columns in column and result-set metadata. This option allows you to set the precision for Decimal columns when using an application that does not support unbounded data types.

Valid Values

x

where:

x

is an integer greater than 0 (zero).

Notes

- The value specified for this option must be greater than or equal to the setting of the Decimal Scale connection option; otherwise, an error is returned when attempting to establish a connection.
- In most scenarios, an error is returned if the size of an Decimal value specified in a statement exceeds the precision determined by this option. However, when executing a Select statement, the driver will return data containing values that are larger than the specified precision.

Default
38

GUI Tab

Schema Map tab

See also

- Decimal Scale on page 97

Decimal Scale

Attribute

DecimalScale (DS)

Purpose

Specifies the maximum scale reported for Decimal columns in column and result-set metadata. This option allows you to set the scale for Decimal columns when using an application that does not support unbounded data types.

Valid Values

x

where:

x

is an integer greater than 0 (zero).

Default
10

Notes

- The value specified for this option cannot exceed the setting of the Decimal Precision connection option; otherwise, an error is returned when attempting to establish a connection.
- In most scenarios, an error is returned if the scale of a Decimal value specified in a statement exceeds the scale determined by this option. However, when executing a Select statement, the driver will return data containing values that have a scale larger than that specified by this option.

GUI Tab

Schema Map tab

See Also

- Decimal Precision on page 96

Description

Attribute

Description (n/a)

Purpose

Specifies an optional long description of a data source. This description is not used as a runtime connection attribute, but does appear in the `ODBC.INI` section of the Registry and in the `odbc.ini` file.

Valid Values

`string`

where:

`string`

is a description of a data source.

Default

None

GUI Tab

General tab

Fetch Size

Attribute

FetchSize (FS)

Purpose

Specifies the number of rows that the driver processes before returning data to the application when executing a Select. This value provides a suggestion to the driver as to the number of rows it should internally process before returning control to the application. The driver may fetch fewer rows to conserve memory when processing exceptionally wide rows.

Valid Values

`0` | *`x`*

where:

x

is a positive integer indicating the number of rows that should be processed.

Behavior

If set to `0`, the driver processes all the rows of the result before returning control to the application. When large data sets are being processed, setting FetchSize to `0` can diminish performance and increase the likelihood of out-of-memory errors.

If set to *x*, the driver limits the number of rows that may be processed for each fetch request before returning control to the application.

Notes

- To optimize throughput and conserve memory, the driver uses an internal algorithm to determine how many rows should be processed based on the width of rows in the result set. Therefore, the driver may process fewer rows than specified by FetchSize when the result set contains exceptionally wide rows. Alternatively, the driver processes the number of rows specified by FetchSize when the result set contains rows of unexceptional width.
- FetchSize can be used to adjust the trade-off between throughput and response time. Smaller fetch sizes can improve the initial response time of the query. Larger fetch sizes can improve overall response times at the cost of additional memory.
- You can use FetchSize to reduce demands on memory and decrease the likelihood of out-of-memory errors. Simply, decrease FetchSize to reduce the number of rows the driver is required to process before returning data to the application.
- The Result Memory Size (ResultMemorySize) connection option can also be used to reduce demands on memory and decrease the likelihood of out-of-memory errors.
- Fetch Size is related to, but different than, Native Fetch Size (NativeFetchSize). Native Fetch Size specifies the number of rows of raw data that the driver fetches from the native data source, while Fetch Size specifies how many of these rows the driver processes before returning control to the application. Processing the data includes converting native data types to SQL data types used by the application. If Fetch Size is greater than Native Fetch Size, the driver may make multiple round trips to the data source to get the requested number of rows before returning control to the application.

Default

`100` (rows)

GUI Tab

Advanced tab

Host Name

Attribute

HostName (HOST)

Purpose

The host name or the IP address of the server to which you want to connect.

Valid Values

host_name | *IP_address*

where:

host_name

is the name of the server to which you want to connect.

IP_address

is the IP address of the server to which you want to connect.

The IP address can be specified in either IPv4 or IPv6 format.

Default

None

GUI Tab

General tab

IANAAppCodePage

UNIX®

Attribute

IANAAppCodePage (IACP)

Purpose

An Internet Assigned Numbers Authority (IANA) value. You must specify a value for this option if your application is not Unicode enabled or if your database character set is not Unicode.

The driver uses the specified IANA code page to convert "W" (wide) functions to ANSI.

The driver and Driver Manager both check for the value of IANAAppCodePage in the following order:

- In the connection string
- In the Data Source section of the system information file (`odbc.ini`)
- In the ODBC section of the system information file (`odbc.ini`)

If the driver does not find an IANAAppCodePage value, the driver uses the default value of 4 (ISO 8859-1 Latin-1).

Valid Values

IANA_code_page

where:

`IANA_code_page`

is one of the valid values listed in "IANAAppCodePage values" in the *Progress DataDirect for ODBC Drivers Reference*. The value must match the database character encoding and the system locale.

Default

4 (ISO 8559-1 Latin-1)

GUI Tab

NA

See Also

Refer to "Internationalization, localization, and Unicode" in the *Progress DataDirect for ODBC Drivers Reference* for details.

Initialization String

Attribute

InitializationString (IS)

Purpose

One or multiple SQL commands to be executed by the driver after it has established the connection to the database and has performed all initialization for the connection. If the execution of a SQL command fails, the connection attempt also fails and the driver returns an error indicating which SQL command or commands failed.

Valid Values

`string`

where:

`string`

is one or multiple SQL commands.

Multiple commands must be separated by semicolons. In addition, if this option is specified in a connection URL, the entire value must be enclosed in parentheses when multiple commands are specified.

Example

When connecting to Cassandra for the first time, the driver queries the Cassandra metadata catalog and caches a view of the catalog in files stored on the client file system. The cached metadata is used in subsequent connections made by the user instead of re-fetching the metadata from Apache Cassandra. To force the driver to refresh the metadata information for every connection, use the Initialization String option to pass the REFRESH MAP command in the connection URL. For example:

```
DSN=Cassandra;UID={CassandraID};PWD=secret;InitializationString=(REFRESH MAP)
```

Notes

- The purpose of this property is to execute commands that the application needs on every connection attempt. Primarily, these commands are used to alter the connect status or to get the latest view of the tables and columns defined in the Cassandra catalog. Therefore, commands specified through this property, such as `SELECT`, will not return result sets.

Default

None

GUI Tab

Advanced tab

JVM Arguments

Attribute

JVMArgs (JVMA)

Purpose

A string that contains the arguments that are passed to the JVM that the driver is starting. The location of the JVM must be specified on the driver library path. For information on setting the location of the JVM in your environment, see:

- Setting the library path environment variable (Windows) on page 30
- Setting the library path environment variable (UNIX and Linux) on page 33.

When specifying the heap size for the JVM, note that the JVM tries to allocate the heap memory as a single contiguous range of addresses in the application’s memory address space. If the application's address space is fragmented so that there is no contiguous range of addresses big enough for the amount of memory specified for the JVM, the driver fails to load, because the JVM cannot allocate its heap. This situation is typically encountered only with 32-bit applications, which have a much smaller application address space. If you encounter problems with loading the driver in an application, try reducing the amount of memory requested for the JVM heap. If possible, switch to a 64-bit version of the application.

Valid Values

`string`

where:

`string`

contains arguments that are defined by the JVM. Values that include special characters or spaces must be enclosed in curly braces { } when used in a connection string.

Example

To set the heap size used by the JVM to 256 MB and the http proxy information, specify:

```
{-Xmx256m -Dhttp.proxyHost=johndoe -Dhttp.proxyPort=808}
```

To set the heap size to 256 MB and configure the JVM for remote debugging, specify:

```
{-Xmx256m -Xrunjdwp:transport=dt_socket, address=9003,server=y,suspend=n -Xdebug}
```

Default

For the 32-bit driver:

`-Xmx256m`

For the 64-bit driver:

`-Xmx1024m`

GUI Tab

SQL Engine tab

JVM Classpath

Attribute

JVMClasspath (JVMC)

Purpose

Specifies the CLASSPATH for the Java Virtual Machine (JVM) used by the driver. The CLASSPATH is the search string the JVM uses to locate the Java jar files the driver needs.

If you do not specify a value, the driver automatically detects the CLASSPATHs for all ODBC drivers installed on your machine and specifies them when launching the JVM.

Valid Values

`string`

where:

`string`

specifies the CLASSPATH. Separate multiple jar files by a semi-colon on Windows platforms and by a colon on Linux and UNIX platforms. CLASSPATH values with multiple jar files must be enclosed in curly braces { } when used in a connection string.

If your process employs multiple drivers that use a JVM, the value of the JVM Classpath for all affected drivers must include an absolute path to all the jar files for drivers used in that process. The value specified for this option must be identical for all drivers. For example, if you are using the Salesforce and Cassandra drivers on Windows, you would specify a value of {`c:\install_dir\java\lib\cassandra.jar;c:\install_dir\java\lib\sforce.jar`} for both drivers. If the value for any of the affected drivers differs from that of the other drivers, the drivers will return an error at connection that the JVM is already running.

Example

On Windows:

```
{.;c:\install_dir\java\lib\cassandra.jar}
```

On UNIX:

```
{.:/home/user1/install_dir/java/lib/cassandra.jar}
```

Default

The default is an empty string, which means that the driver automatically detects the CLASSPATHs for all ODBC drivers installed on your machine and specifies them when launching the JVM.

GUI Tab

SQL Engine tab

Keyspace Name

Attribute

KeyspaceName (KSN)

Purpose

Specifies the name of the keyspace to which you want to connect. This value is also used as the default qualifier for unqualified table names in queries.

Valid Values

keyspace_name

where:

keyspace_name

is the name of a valid keyspace. If the driver cannot find the specified keyspace, the connection fails; however, if you do not specify a value, the default is the keyspace defined by the system administrator for each user.

Notes

- If KeyspaceName is not specified, Cassandra's internal keyspace name `system` will be used.

Default

`system`

GUI Tab

General tab

Log Config File

Attribute

LogConfigFile (LCF)

Purpose

Specifies the filename of the configuration file used to initialize the driver logging mechanism.

Valid Values

string

where:

string

is the relative or fully qualified path of the configuration file used to initialize the driver logging mechanism. If the specified file does not exist, the driver continues searching for an appropriate configuration file as described in "Logging for Java components".

Notes

- If the driver cannot locate the specified file when establishing the connection, it looks for a properties file named *user_name*`.logging.properties` in the application working directory. If the driver cannot find the file in the application directory, the driver looks for a file named `ddlogging.properties` in the current working directory. If the driver cannot find the file, the driver abandons its attempt to load a properties file and connects.

Default

The driver looks for the file named `ddlogging.properties` in the current working directory to load for all connections. If the driver is operating in Server mode, the driver uses the `ddlogging.properties` file that is stored in the application working directory.

GUI Tab

Advanced tab

See Also

- For more information, refer to "Logging for Java components" in the *Progress DataDirect for ODBC Drivers Reference*.

Login Timeout

Attribute

LoginTimeout (LT)

Purpose

The number of seconds the driver waits for a connection to be established before returning control to the application and generating a timeout error. To override the value that is set by this connection option for an individual connection, set a different value in the SQL_ATTR_LOGIN_TIMEOUT connection attribute using the SQLSetConnectAttr() function.

Valid Values

`-1` | `0` | *x*

where:

x

is a positive integer that represents a number of seconds.

Behavior

If set to `-1`, the connection request does not time out. The driver silently ignores the SQL_ATTR_LOGIN_TIMEOUT attribute.

If set to `0`, the connection request does not time out, but the driver responds to the SQL_ATTR_LOGIN_TIMEOUT attribute.

If set to *x*, the connection request times out after the specified number of seconds unless the application overrides this setting with the SQL_ATTR_LOGIN_TIMEOUT attribute.

Default

`15`

GUI Tab

Advanced tab

Native Fetch Size

Attribute

NativeFetchSize (NFS)

Purpose

Specifies the number of rows of data the driver attempts to fetch from the native data source on each request submitted to the server.

Valid Values

`0` | *x*

where:

x

is a positive integer that defines the number of rows.

Behavior

If set to `0`, the driver requests that the server return all rows for each request submitted to the server. Block fetching is not used.

If set to *x*, the driver attempts to fetch up to a maximum of the specified number of rows on each request submitted to the server.

Notes

Native Fetch Size is related to, but different than, Fetch Size. Native Fetch Size specifies the number of rows of raw data that the driver fetches from the native data source, while Fetch Size specifies how many of these rows the driver processes before returning control to the application. Processing the data includes converting native data types to SQL data types used by the application. If Fetch Size is greater than Native Fetch Size, the driver may make multiple round trips to the data source to get the requested number of rows before returning control to the application.

Default

`10000` (rows)

GUI Tab

Advanced tab

Password

Attribute

Password (PWD)

Purpose

Specifies the password to use to connect to your database. Contact your system administrator to obtain your password.

Important: Setting the password using a data source is not recommended. The data source persists all options, including the Password option, in clear text.

Valid Values

`password`

where:

`password`

is a valid password. The password is case-sensitive.

Default

None

GUI Tab

Logon Dialog

Port Number

Attribute

PortNumber (PORT)

Purpose

Specifies the port number of the server listener.

Valid Values

port_number

where:

port_number

is the port number of the server listener. Check with your database administrator for the correct number.

Default

9042

GUI Tab

General tab

Read Consistency

Attribute

ReadConsistency (RC)

Purpose

Specifies how many replicas must respond to a read request before returning data to the client application.

Valid Values

1 | 2 | 3 | 4 | 5 | 6 | 8 | 9 | 10

Behavior

If set to 1 (one), data is returned from the closest replica. This setting provides the highest availability, but increases the likelihood of stale data being read.

If set to 2 (two), data is returned from two of the closest replicas.

If set to 3 (three), data is returned from three of the closest replicas.

If set to 4 (quorum), data is returned after a quorum of replicas has responded from any data center.

If set to 5 (all), data is returned to the application after all replicas have responded. This setting provides the highest consistency and lowest availability.

If set to 6 (local_quorum), data is returned after a quorum of replicas in the same data center as the coordinator node has responded. This setting avoids latency of inter-data center communication.

If set to 8 (serial), the data is read without proposing a new addition or update. Uncommitted transactions are committed as part of the read.

If set to 9 (local_serial), the data within a data center is read without proposing a new addition or update. Uncommitted transactions within the data center are committed as part of the read.

If set to 10 (local_one), data is returned from the closest replica in the local data center.

Notes

- If the server does not support the Read Consistency value specified, the connection attempt fails and the driver returns a consistency level error.
- Refer to Apache Cassandra documentation for more information about configuring consistency levels, including usage scenarios.

Default

4 (quorum)

GUI Tab

Advanced tab

Read Only

Attribute

ReadOnly (RO)

Purpose

Specifies whether the connection has read-only access to the data source.

Valid Values

0 | 1

Behavior

If set to 1 (Enabled), the connection has read-only access. The following commands are the only commands that you can use when a connection is read-only:

The driver returns an error if any other command is executed.

If set to 0 (Disabled), the connection is opened for read/write access, and you can use all commands supported by the product.

Default

0 (Disabled)

GUI Tab

Advanced tab

Report Codepage Conversion Errors

Attribute

ReportCodepageConversionErrors (RCCE)

Purpose

Specifies, in certain scenarios, how the driver handles code page conversion errors that occur when a character cannot be converted from one character set to another.

An error message or warning can occur if an ODBC call causes a conversion error, or if an error occurs during code page conversions to and from the database or to and from the application, for example: `Code page conversion error encountered`.

Valid Values

0 | 1 | 2

Behavior

If set to 0 (Ignore Errors), the driver substitutes 0x1A for each character that cannot be converted and does not return a warning or error.

If set to 1 (Return Error), the driver returns an error instead of substituting 0x1A for unconverted characters.

If set to 2 (Return Warning), the driver substitutes 0x1A for each character that cannot be converted and returns a warning.

Default

0 (Ignore Errors)

GUI Tab

Advanced tab

Result Memory Size

Attribute

ResultMemorySize (RMS)

Purpose

Specifies the maximum size, in megabytes, of an intermediate result set that the driver holds in memory. When this threshold is reached, the driver writes a portion of the result set to disk in temporary files.

Valid Values

`-1` | `0` | *x*

where:

x

is a positive integer.

Behavior

If set to `-1`, the maximum size of an intermediate result set that the driver holds in memory is determined by a percentage of the max Java heap size. When this threshold is reached, the driver writes a portion of the result set to disk.

If set to `0`, the SQL engine holds intermediate results in memory regardless of size. Setting Result Memory Size to `0` can increase performance for any result set that can easily fit within the JVM's free heap space, but can decrease performance for any result set that can barely fit within the JVM's free heap space.

If set to *x*, the SQL engine holds intermediate results in memory that are no larger than the size specified. When this threshold is reached, the driver writes a portion of the result set to disk.

Notes

- By default, Result Memory Size is set to `-1`. When set to `-1`, the maximum size of an intermediate result that the driver holds in memory is a percentage of the max Java heap size. When processing large sets of data, out-of-memory errors can occur when the size of the result set exceeds the available memory allocated to the JVM. In this scenario, you can tune Result Memory Size to suit your environment. To begin, set Result Memory Size equal to the max Java heap size divided by 4. Proceed by decreasing this value until out-of-memory errors are eliminated. As a result, the maximum size of an intermediate result set the driver holds in memory will be reduced, and some portion of the result set will be written to disk. Be aware that while writing to disk reduces the risk of out-of-memory errors, it also negatively impacts performance. For optimal performance, decrease this value only to a size necessary to avoid errors.
- You can also address memory and performance concerns by adjusting the max Java heap size using the JVM Arguments connection option. By increasing the max Java heap size, you increase the amount of data the driver accumulates in memory. This can reduce the likelihood of out-of-memory errors and improve performance by ensuring that result sets fit easily within the JVM's free heap space. In addition, when a limit is imposed by setting Result Memory Size to `-1`, increasing the max Java heap size can improve performance by reducing the need to write to disk, or removing it altogether.
- The Fetch Size connection option can also be used to reduce demands on the driver's internal memory and decrease the likelihood of out-of-memory errors.

Default

`-1`

GUI Tab

Advanced tab

See also

- Performance considerations on page 70
- Fetch Size on page 98
- JVM Arguments on page 102

Schema Map

Attribute

SchemaMap (SMP)

Purpose

Specifies the name and location of the configuration file used to create the relational map of native data. The driver looks for this file when connecting to the server. If the file does not exist, the driver creates one.

Valid Values

string

where:

string

is the absolute path and filename of the configuration file, including the `.config` extension. For example, if SchemaMap is set to a value of `C:\Users\Default\AppData\Local\Progress\DataDirect\Cassandra_Schema\MyServer.config`, the driver either creates or looks for the configuration file `MyServer.config` in the directory `C:\Users\Default\AppData\Local\Progress\DataDirect\Cassandra_Schema\`.

Notes

- When connecting to a server, the driver looks for the SchemaMap configuration file. If the configuration file does not exist, the driver creates a SchemaMap configuration file using the name and location you have provided. If you do not provide a name and location for a SchemaMap configuration file, the driver creates it using default values.
- The driver uses the path specified in this connection option to store additional internal files.
- You can refresh the internal files related to an existing relational view of your data by using the SQL extension Refresh Map. Refresh Map queries the Cassandra catalog tables and updates your internal files accordingly.

Default

The default is determined by the environment. The driver attempts to create the files in a subdirectory of the first available directory in the following order:

- Windows
  - DD_HOME environment variable
  - LOCALAPPDATA environment variable
  - APPDATA environment variable
  - User’s home directory

For Windows, the file path takes the following format:

available_location\Progress\DataDirect\Cassandra_Schema*user_name*.config

- UNIX/Linux
  - DD_HOME environment variable
  - User's home directory

  For UNIX/Linux, the file path takes the following format:

  available_location/progress/datadirect/Cassandra_schema/*user_name*.config

GUI Tab

Schema Map tab

Server Port Number

Attribute

ServerPortNumber (SPN)

Purpose

Specifies a valid port on which the SQL engine listens for requests from the driver.

Valid Values

port_name

where:

port_name

is the port number of the server listener. Check with your system administrator for the correct number.

Notes

- This option is ignored unless SQL Engine Mode is set to `0` (Auto) or `1` (Server).

Default

For the 32-bit driver:

`19934`

For the 64-bit driver:

`19933`

GUI Tab

SQL Engine tab

SQL Engine Mode

Attribute

SQLEngineMode (SEM)

Purpose

Specifies whether the driver's SQL engine runs in the same process as the driver (direct mode) or runs in a process that is separate from the driver (server mode). You must be an administrator to modify the server mode configuration values, and to start or stop the SQL engine service.

Valid Values

`0` | `1` | `2`

Behavior

If set to `0` (Auto), the SQL engine attempts to run in server mode first; however, if server mode is unavailable, it runs in direct mode. To use server mode with this value, you must start the SQL engine service before using the driver (see "Starting the SQL engine server" for more information).

If set to `1` (Server), the SQL engine runs in server mode. The SQL engine operates in a separate process from the driver within its own JVM. You must start the SQL Engine service before using the driver (see "Starting the SQL engine server" for more information).

If set to `2` (Direct), the SQL engine runs in direct mode. The driver and its SQL engine run in a single process within the same JVM.

Notes

- **Important**: Changes you make to the server mode configuration affect all DSNs sharing the service.

Default

For Windows:

`0` (Auto)

For UNIX/Linux:

`2` (Direct)

GUI Tab

SQL Engine tab

See Also

Starting the SQL engine server on Windows on page 74

Transaction Mode

Attribute

TransactionMode (TM)

Purpose

Specifies how the driver handles manual transactions.

Valid Values

`0` | `1`

Behavior

If set to `1` (Ignore), the data source does not support transactions and the driver always operates in auto-commit mode. Calls to set the driver to manual commit mode and to commit transactions are ignored. Calls to rollback a transaction cause the driver to return an error indicating that no transaction is started. Metadata indicates that the driver supports transactions and the ReadUncommitted transaction isolation level.

If set to `0` (No Transactions), the data source and the driver do not support transactions. Metadata indicates that the driver does not support transactions.

Default

`0` (No Transactions)

GUI Tab

Advanced tab

User Name

Attribute

LogonID (UID)

Purpose

The default user ID that is used to connect to your database. Your ODBC application may override this value or you may override it in the logon dialog box or connection string.

Valid Values

`userid`

where:

`userid`

is a valid user ID with permissions to access the database.

GUI Tab

Security tab

Varchar Size

Attribute

VarcharSize (VCS)

Purpose

Specifies the precision reported for Varchar columns in column and result-set metadata. This option allows you to set the precision for Varchar columns when using an application that does not support unbounded data types.

Valid Values

x

where:

x

is an integer greater than 0 (zero).

Default

`4000`

Notes

- In most scenarios, an error is returned if the size of a Varchar value specified in a statement exceeds the precision determined by this option. However, when executing a Select statement, the driver will return data containing values that are larger than the specified precision.

GUI Tab

Schema Map tab

Varint Precision

Attribute

VarintPrecision (VP)

Purpose

Specifies the precision reported for Varint columns in column and result-set metadata. This option allows you to set a the precision for Varint columns when using an application that does not support unbounded data types.

Valid Values

x

where:

x

is an integer greater than 0 (zero).

Default

38

Notes

- In most scenarios, an error is returned if the size of a Varint value specified in a statement exceeds the precision determined by this option. However, when executing a Select statement, the driver returns data containing values that are larger than the specified precision.

GUI Tab

Schema Map tab

Write Consistency

Attribute

WriteConsistency (WC)

Purpose

Determines the number of replicas on which the write must succeed before returning an acknowledgment to the client application.

Valid Values

`0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9` | `10`

Behavior

If set to `0` (any), a write must succeed on at least one node. Even if all replica nodes for the given partition key are down, the write can succeed after a hinted handoff has been written. This setting provides the lowest consistency and highest availability.

If set to `1` (one), a write must succeed on at least one replica node.

If set to `2` (two), a write must succeed on at least two replica nodes.

If set to `3` (three), a write must succeed on at least three replica nodes.

If set to `4` (quorum), a write must succeed on a quorum of replica nodes.

If set to `5` (all), a write must succeed on all replica nodes in the cluster for that partition key. This setting provides the highest consistency and lowest availability.

If set to `6` (local_quorum), a write must succeed on a quorum of replica nodes in the same data center as the coordinator node. This setting avoids latency of inter-data center communication.

If set to 7 (each_quorum), a write must succeed on a quorum of replica nodes across a data center.

(Cassandra 2.x only) If set to 8 (serial), the driver prevents unconditional updates to achieve linearizable consistency for lightweight transactions.

(Cassandra 2.x only) If set to 9 (local_serial), the driver prevents unconditional updates to achieve linearizable consistency for lightweight transactions within the data center.

If set to 10 (local_one), a write must succeed on at least one replica node in the local data center.

Notes

- An update operation can result in a consistency level error if the server does not support the WriteConsistency value specified.
- Refer to Apache Cassandra documentation for more information about configuring consistency levels, including usage scenarios.

Default

4 (quorum)

GUI Tab

Advanced tab

6

Supported SQL functionality

The driver provides support for SQL statements and extensions described in this section. SQL extensions are denoted by an (EXT) in the topic title.

For details, see the following topics:

- Data definition language (DDL)
- Delete
- Insert
- Refresh Map (EXT)
- Select
- Update
- SQL expressions
- Subqueries

Data definition language (DDL)

The driver supports data store-specific DDL through the Native and Refresh escape clauses. See "Native and Refresh escape clauses" for details.

Native and Refresh escape clauses

The driver supports `ODBC-native-escape` and `ODBC-refresh-schema-escape` clauses to embed data store-specific commands in SQL-92 statements. The `ODBC-native-escape` clause allows you to execute native commands directly through the client application, while the `ODBC-refresh-schema-escape` clause is used to incorporate any changes introduced by the `ODBC-native-escape` clause into the driver's relational map of the data.

Note: The Native and Refresh escape clauses are mainly intended for the execution of DDL commands, such as ALTER, CREATE, and DROP. A returning clause for the Native escape is not currently supported by the driver. Therefore, results cannot be retrieved using the Native escape clause.

The `ODBC-native-escape` clause is defined as follows:

`ODBC-native-escape ::= ODBC-esc-initiator native (`*`command_text`*`)}`

where:

`command_text`

is a data store-specific command.

The `ODBC-refresh-schema-escape` clause is defined as follows:

`ODBC-refresh-schema-escape ::= ODBC-esc-initiator refresh ODBC-esc-terminator`

The following example shows the execution of two data store-specific commands with a refresh of the driver's relational map of the data. Note that each Native escape clause must have its own execute. The Refresh escape, however, can be used in the same execute statement as the Native escape.

```
SQLExecDirect(
   pstmt,
   "{native (CREATE TABLE emp (empid int primary key, title varchar))}",
   SQL_NTS);
SQLExecDirect(
   pstmt,
   "{native (CREATE TABLE dept (deptid int primary key, city varchar))}{refresh}",
   SQL_NTS);
```

See also

Supported SQL functionality on page 119

Delete

Purpose

The Delete statement is used to delete rows from a table.

Syntax

`DELETE FROM `*`table_name`*` [WHERE `*`search_condition`*`]`

where:

table_name

specifies the name of the table from which you want to delete rows.

search_condition

is an expression that identifies which rows to delete from the table.

Notes

- The Where clause determines which rows are to be deleted. Without a Where clause, all rows of the table are deleted, but the table is left intact. See "Where Clause" for information about the syntax of Where clauses. Where clauses can contain subqueries.

Example A

This example shows a Delete statement on the emp table.

```
DELETE FROM EMP WHERE EMP_ID = 'E10001'
```

Each Delete statement removes every record that meets the conditions in the Where clause. In this case, every record having the employee ID `E10001` is deleted. Because employee IDs are unique in the employee table, at most, one record is deleted.

Example B

This example shows using a subquery in a Delete clause.

```
DELETE FROM EMP WHERE DEPT_ID = (SELECT DEPT_ID FROM DEPT WHERE DEPT_NAME =
'Marketing')
```

The records of all employees who belong to the department named Marketing are deleted.

Notes

- Delete is supported for primitive types and non-nested complex types. See "Complex Type Normalization" for details.
- To enable Insert, Update, and Delete, set the Read Only (ReadOnly) connection option to `0` (Disabled).
- A Where clause can be used to restrict which rows are deleted.

See also

- Where clause on page 129
- Complex type normalization on page 18

Insert

Purpose

The Insert statement is used to add new rows to a local table. You can specify either of the following options:

- List of values to be inserted as a new row
- Select statement that copies data from another table to be inserted as a set of new rows

In Cassandra, Inserts are in effect Upserts. When an Insert is performed on a row that already exists, the row will be updated.

Syntax

```
INSERT INTO table_name [(column_name[,column_name]...)] {VALUES (expression
[,expression]...) | select_statement}
```

table_name

is the name of the table in which you want to insert rows.

column_name

is optional and specifies an existing column. Multiple column names (a column list) must be separated by commas. A column list provides the name and order of the columns, the values of which are specified in the Values clause. If you omit a *column_name* or a column list, the value expressions must provide values for all columns defined in the table and must be in the same order that the columns are defined for the table. Table columns that do not appear in the column list are populated with the default value, or with NULL if no default value is specified.

expression

is the list of expressions that provides the values for the columns of the new record. Typically, the expressions are constant values for the columns. Character string values must be enclosed in single quotation marks (’). See "Literals" for more information.

select_statement

is a query that returns values for each *column_name* value specified in the column list. Using a Select statement instead of a list of value expressions lets you select a set of rows from one table and insert it into another table using a single Insert statement. The Select statement is evaluated before any values are inserted. This query cannot be made on the table into which values are inserted. See "Select" for information about Select statements.

Notes

- Insert is supported for primitive types and non-nested complex types. See "Complex type normalization" for details.
- The driver supports an insert on a child table prior to an insert on a parent table, circumventing referential integrity constraints associated with traditional RDBMS. To maintain integrity between parent and child tables, it is recommended that an insert first be performed on the parent table for each foreign key value added to the child. If such an insert is not first performed, the driver automatically inserts a row into the parent table that contains only the primary key values and NULL values for all non-primary key columns. See "Complex type normalization" for details.
- To enable Insert, Update, and Delete, set the Read Only (ReadOnly) connection option to `0` (Disabled).

See also

- Literals on page 137
- Select on page 123
- Complex type normalization on page 18

Refresh Map (EXT)

Purpose

The REFRESH MAP statement adds newly discovered objects to your relational view of native data. It also incorporates any configuration changes made to your relational view by reloading the schema map configuration file.

Syntax

```
REFRESH MAP
```

Notes

REFRESH MAP is an expensive query since it involves the discovery of native data.

Select

Purpose

Use the Select statement to fetch results from one or more tables.

Syntax

```
SELECT select_clause from_clause
[where_clause]
[groupby_clause]
[having_clause]
[{UNION [ALL | DISTINCT] |
 {MINUS [DISTINCT] | EXCEPT [DISTINCT]} |
 INTERSECT [DISTINCT]} select_statement]
[limit_clause]
```

where:

select_clause

specifies the columns from which results are to be returned by the query. See "Select clause" for a complete explanation.

from_clause

specifies one or more tables on which the other clauses in the query operate. See "From clause" for a complete explanation.

where_clause

is optional and restricts the results that are returned by the query. See "Where clause" for a complete explanation.

groupby_clause

is optional and allows query results to be aggregated in terms of groups. See "Group By clause" for a complete explanation.

having_clause

is optional and specifies conditions for groups of rows (for example, display only the departments that have salaries totaling more than $200,000). See "Having clause" for a complete explanation.

UNION

is an optional operator that combines the results of the left and right Select statements into a single result. See "Union operator" for a complete explanation.

INTERSECT

is an optional operator that returns a single result by keeping any distinct values from the results of the left and right Select statements. See "Intersect operator" for a complete explanation.

EXCEPT | MINUS

are synonymous optional operators that returns a single result by taking the results of the left Select statement and removing the results of the right Select statement. See "Except and Minus operators" for a complete explanation.

orderby_clause

is optional and sorts the results that are returned by the query. See "Order By clause" for a complete explanation.

limit_clause

is optional and places an upper bound on the number of rows returned in the result. See "Limit clause" for a complete explanation.

See also

Select clause on page 125
From clause on page 127
Where clause on page 129
Group By clause on page 129
Having clause on page 130
Union operator on page 131
Intersect operator on page 131
Except and Minus operators on page 132
Order By clause on page 133
Limit clause on page 134

Select clause

Purpose

Use the Select clause to specify with a list of column expressions that identify columns of values that you want to retrieve or an asterisk (*) to retrieve the value of all columns.

Syntax

```
SELECT [{LIMIT offset number | TOP number}] [ALL | DISTINCT] {* | column_expression
 [[AS] column_alias] [,column_expression [[AS] column_alias], ...]}
```

where:

`LIMIT offset number`

creates the result set for the Select statement first and then discards the first number of rows specified by *`offset`* and returns the number of remaining rows specified by *`number`*. To not discard any of the rows, specify `0` for *`offset`*, for example, `LIMIT 0` *`number`*. To discard the first *`offset`* number of rows and return all the remaining rows, specify `0` for *`number`*, for example, `LIMIT` *`offset`*`0`.

`TOP` *`number`*

is equivalent to `LIMIT 0`*`number`*.

`column_expression`

can be simply a column name (for example, `last_name`). More complex expressions may include mathematical operations or string manipulation (for example, `salary * 1.05`). See "SQL expressions" for details. *`column_expression`* can also include aggregate functions. See "Aggregate functions" for details.

`column_alias`

can be used to give the column a descriptive name. For example, to assign the alias department to the column dep:

```
SELECT DEP AS DEPARTMENT FROM EMP
```

`DISTINCT`

eliminates duplicate rows from the result of a query. This operator can precede the first column expression. For example:

```
SELECT DISTINCT DEP FROM EMP
```

Notes

- Separate multiple column expressions with commas (for example, `SELECT LAST_NAME, FIRST_NAME, HIRE_DATE`).
- Column names can be prefixed with the table name or table alias. For example, `SELECT EMP.LAST_NAME or E.LAST_NAME`, where `E` is the alias for the table `EMP`.
- NULL values are not treated as distinct from each other. The default behavior is that all result rows be returned, which can be made explicit with the keyword `ALL`.

See also

SQL expressions on page 136
Aggregate functions on page 126

Aggregate functions

Aggregate functions can also be a part of a Select clause. Aggregate functions return a single value from a set of rows. An aggregate can be used with a column name (for example, `AVG(SALARY)`) or in combination with a more complex column expression (for example, `AVG(SALARY * 1.07)`). The column expression can be preceded by the `DISTINCT` operator. The `DISTINCT` operator eliminates duplicate values from an aggregate expression.

The following table lists supported aggregate functions.

Table 17: Aggregate Functions

Aggregate	Returns
AVG	The average of the values in a numeric column expression. For example, `AVG(SALARY)` returns the average of all salary column values.
COUNT	The number of values in any field expression. For example, `COUNT(NAME)` returns the number of name values. When using `COUNT` with a field name, `COUNT` returns the number of non-NULL column values. A special example is `COUNT(*)`, which returns the number of rows in the set, including rows with NULL values.
MAX	The maximum value in any column expression. For example, `MAX(SALARY)` returns the maximum salary column value.
MIN	The minimum value in any column expression. For example, `MIN(SALARY)` returns the minimum salary column value.
SUM	The total of the values in a numeric column expression. For example, `SUM(SALARY)` returns the sum of all salary column values.

Example A

In the following example, only distinct last name values are counted. The default behavior is that all duplicate values be returned, which can be made explicit with `ALL`.

```
COUNT (DISTINCT LAST_NAME)
```

Example B

The following example uses the `COUNT`, `MAX`, and `AVG` aggregate functions:

```
SELECT
     COUNT(AMOUNT) AS numOpportunities,
     MAX(AMOUNT) AS maxAmount,
     AVG(AMOUNT) AS avgAmount
FROM OPPORTUNITY O INNER JOIN USER U
     ON O.OWNERID = U.ID
WHERE O.ISCLOSED = 'false' AND
     U.NAME = 'MyName'
```

From clause

Purpose

The From clause indicates the tables to be used in the Select statement.

Syntax

```
FROM table_name [table_alias] [,...]
```

where:

table_name

is the name of a table or a subquery. Multiple tables define an implicit inner join among those tables. Multiple table names must be separated by a comma. For example:

```
SELECT * FROM EMP, DEP
```

Subqueries can be used instead of table names. Subqueries must be enclosed in parentheses. See "Subquery in a From clause" for an example.

table_alias

is a name used to refer to a table in the rest of the Select statement. When you specify an alias for a table, you can prefix all column names of that table with the table alias.

Example

The following example specifies two table aliases, E for EMP and D for DEP:

```
SELECT E.NAME, D.DEPTNAME
FROM EMP E, DEP D
WHERE E.DEPTID = D.ID
```

table_alias is a name used to refer to a table in the rest of the Select statement. When you specify an alias for a table, you can prefix all column names of that table with the table alias. For example, given the table specification:

```
FROM EMP E
```

you may refer to the LAST_NAME field as E.LAST_NAME. Table aliases must be used if the Select statement joins a table to itself. For example:

```
SELECT * FROM EMP E, EMP F WHERE E.MGR_ID = F.EMP_ID
```

The equal sign (=) includes only matching rows in the results.

See also

Subquery in a From clause on page 128

Outer join escape sequences

Purpose

The SQL-92 left, right, and full outer join syntax is supported.

Syntax

```
{oj outer-join}
```

where *outer-join* is

```
table-reference {LEFT | RIGHT | FULL} OUTER JOIN {table-reference | outer-join} ON
search-condition
```

where *table-reference* is a database table name, and *search-condition* is the join condition you want to use for the tables.

For example:

```
SELECT CUSTOMERS.CUSTID, CUSTOMERS.NAME, ORDERS.ORDERID, ORDERS.STATUS FROM {oj CUSTOMERS
 LEFT OUTER JOIN ORDERS ON CUSTOMERS.CUSTID=ORDERS.CUSTID} WHERE ORDERS.STATUS='OPEN'
```

The following outer join escape sequences are supported:

- Left outer joins
- Right outer joins
- Full outer joins
- Nested outer joins

Join in a From clause

Purpose

You can use a Join as a way to associate multiple tables within a Select statement. Joins may be either explicit or implicit. For example, the following is the example from the previous section restated as an explicit inner join:

```
SELECT * FROM EMP INNER JOIN DEP ON ID=EMPID
SELECT E.NAME, D.DEPTNAME
FROM EMP E INNER JOIN DEP D ON E.DEPTID = D.ID;
```

whereas the following is the same statement as an implicit inner join:

```
SELECT * FROM EMP, DEP WHERE EMP.DEPTID=DEP.ID
```

Syntax

```
FROM table_name {RIGHT OUTER | INNER | LEFT OUTER | CROSS | FULL OUTER} JOIN table.key
 ON search-condition
```

Example

In this example, two tables are joined using `LEFT OUTER JOIN`. `t1`, the first table named includes nonmatching rows.

```
SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.KEY = T2.KEY
```

If you use a `CROSS JOIN`, no `ON` expression is allowed for the join.

Subquery in a From clause

Subqueries can be used in the From clause in place of table references (*table_name*).

Example

```
SELECT * FROM (SELECT * FROM EMP WHERE SAL > 10000) NEW_EMP, DEPT WHERE
NEW_EMP.DEPTNO = DEPT.DEPTNO
```

See also

For more information about subqueries, see Subqueries on page 144.

Where clause

Purpose

Specifies the conditions that rows must meet to be retrieved.

Syntax

```
WHERE expr1 rel_operator expr2
```

where:

expr1

is either a column name, literal, or expression.

expr2

is either a column name, literal, expression, or subquery. Subqueries must be enclosed in parentheses.

rel_operator

is the relational operator that links the two expressions.

Example

The following Select statement retrieves the first and last names of employees that make at least $20,000.

```
SELECT LAST_NAME, FIRST_NAME FROM EMP WHERE SALARY >= 20000
```

See also

- SQL expressions on page 136
- Subqueries on page 144
- Operation timeouts on page 80

Group By clause

Purpose

Specifies the names of one or more columns by which the returned values are grouped. This clause is used to return a set of aggregate values.

Syntax

```
GROUP BY column_expression [,...]
```

where:

column_expression

is either a column name or a SQL expression. Multiple values must be separated by a comma. If *column_expression* is a column name, it must match one of the column names specified in the Select clause. Also, the Group By clause must include all non-aggregate columns specified in the Select list.

Example

The following example totals the salaries in each department:

```
SELECT DEPT_ID, sum(SALARY) FROM EMP GROUP BY DEPT_ID
```

This statement returns one row for each distinct department ID. Each row contains the department ID and the sum of the salaries of the employees in the department.

See also

- SQL expressions on page 136
- Subqueries on page 144

Having clause

Purpose

Specifies conditions for groups of rows (for example, display only the departments that have salaries totaling more than $200,000). This clause is valid only if you have already defined a Group By clause.

Syntax

HAVING *expr1 rel_operator expr2*

where:

expr1 | expr2

can be column names, constant values, or expressions. These expressions do not have to match a column expression in the Select clause. See "SQL expressions" for details regarding SQL expressions.

rel_operator

is the relational operator that links the two expressions.

Example

The following example returns only the departments that have salaries totaling more than $200,000:

```
SELECT DEPT_ID, sum(SALARY) FROM EMP GROUP BY DEPT_ID HAVING sum(SALARY) > 200000
```

See also

- SQL expressions on page 136
- Subqueries on page 144

Union operator

Purpose

Combines the results of two Select statements into a single result. The single result is all the returned rows from both Select statements. By default, duplicate rows are not returned. To return duplicate rows, use the All keyword (`UNION ALL`).

Syntax

```
select_statement
UNION [ALL | DISTINCT] | {MINUS [DISTINCT] | EXCEPT [DISTINCT]} | INTERSECT
[DISTINCT]select_statement
```

Notes

- When using the Union operator, the Select lists for each Select statement must have the same number of column expressions with the same data types and must be specified in the same order.

Example A

The following example has the same number of column expressions, and each column expression, in order, has the same data type.

```
SELECT LAST_NAME, SALARY, HIRE_DATE FROM EMP
UNION
SELECT NAME, PAY, BIRTH_DATE FROM PERSON
```

Example B

The following example is *not* valid because the data types of the column expressions are different (`SALARY FROM EMP` has a different data type than `LAST_NAME FROM RAISES`). This example does have the same number of column expressions in each Select statement but the expressions are not in the same order by data type.

```
SELECT LAST_NAME, SALARY FROM EMP
UNION
SELECT SALARY, LAST_NAME FROM RAISES
```

Intersect operator

Purpose

Intersect operator returns a single result set. The result set contains rows that are returned by both Select statements. Duplicates are returned unless the Distinct operator is added.

Syntax

```
select_statement
INTERSECT [DISTINCT]
select_statement
```

where:

`DISTINCT`

eliminates duplicate rows from the results.

Notes

- When using the Intersect operator, the Select lists for each Select statement must have the same number of column expressions with the same data types and must be specified in the same order.

Example A

The following example has the same number of column expressions, and each column expression, in order, has the same data type.

```
SELECT LAST_NAME, SALARY, HIRE_DATE FROM EMP
INTERSECT [DISTINCT]
SELECT NAME, PAY, BIRTH_DATE FROM PERSON
```

Example B

The following example is *not* valid because the data types of the column expressions are different (`salary FROM emp` has a different data type than `last_name FROM raises`). This example does have the same number of column expressions in each Select statement but the expressions are not in the same order by data type.

```
SELECT LAST_NAME, SALARY FROM EMP
INTERSECT
SELECT SALARY, LAST_NAME FROM RAISES
```

Except and Minus operators

Purpose

Return the rows from the left Select statement that are not included in the result of the right Select statement.

Syntax

```
select_statement
{EXCEPT [DISTINCT] | MINUS [DISTINCT]}
select_statement
```

where:

`DISTINCT`

eliminates duplicate rows from the results.

Notes

- When using one of these operators, the Select lists for each Select statement must have the same number of column expressions with the same data types and must be specified in the same order.

Example A

The following example has the same number of column expressions, and each column expression, in order, has the same data type.

```
SELECT LAST_NAME, SALARY, HIRE_DATE FROM EMP
EXCEPT
SELECT NAME, PAY, BIRTH_DATE FROM PERSON
```

Example B

The following example is *not* valid because the data types of the column expressions are different (`SALARY FROM EMP` has a different data type than `LAST_NAME FROM RAISES`). This example does have the same number of column expressions in each Select statement but the expressions are not in the same order by data type.

```
SELECT LAST_NAME, SALARY FROM EMP
EXCEPT
SELECT SALARY, LAST_NAME FROM RAISES
```

Order By clause

Purpose

The Order By clause specifies how the rows are to be sorted.

Syntax

`ORDER BY` *`sort_expression`* `[DESC | ASC] [,...]`

where:

`sort_expression`

is either the name of a column, a column alias, a SQL expression, or the positioned number of the column or expression in the select list to use.

The default is to perform an ascending (`ASC`) sort.

Example

To sort by `LAST_NAME` and then by `FIRST_NAME`, you could use either of the following Select statements:

```
SELECT EMP_ID, LAST_NAME, FIRST_NAME FROM EMP
ORDER BY LAST_NAME, FIRST_NAME
```

or

```
SELECT EMP_ID, LAST_NAME, FIRST_NAME FROM EMP
ORDER BY 2,3
```

In the second example, `LAST_NAME` is the second item in the Select list, so `ORDER BY 2,3` sorts by `LAST_NAME` and then by `FIRST_NAME`.

See also

SQL expressions on page 136

Limit clause

Purpose

Places an upper bound on the number of rows returned in the result.

Syntax

`LIMIT *number_of_rows* [OFFSET *offset_number*]`

where:

number_of_rows

specifies a maximum number of rows in the result. A negative number indicates no upper bound.

OFFSET

specifies how many rows to skip at the beginning of the result set. *offset_number* is the number of rows to skip.

Notes

- In a compound query, the Limit clause can appear only on the final Select statement. The limit is applied to the entire query, not to the individual Select statement to which it is attached.

Example

The following example returns a maximum of 20 rows.

```
SELECT LAST_NAME, FIRST_NAME FROM EMP WHERE SALARY > 20000 ORDER BY DEPT_IDC LIMIT
20
```

Update

Purpose

An Update statement changes the value of columns in the selected rows of a table.

Syntax

`UPDATE *table_name* SET *column_name* = *expression*`

`[, *column_name* = *expression*] [WHERE *conditions*]`

table_name

is the name of the table for which you want to update values.

column_name

is the name of a column, the value of which is to be changed. Multiple column values can be changed in a single statement.

expression

is the new value for the column. The expression can be a constant value or a subquery that returns a single value. Subqueries must be enclosed in parentheses.

Example A

The following example changes every record that meets the conditions in the Where clause. In this case, the salary and exempt status are changed for all employees having the employee ID `E10001`. Because employee IDs are unique in the emp table, only one record is updated.

```
UPDATE EMP SET SALARY=32000, EXEMPT=1
WHERE EMP_ID = 'E10001'
```

Example B

The following example uses a subquery. In this example, the salary is changed to the average salary in the company for the employee having employee ID `E10001`.

```
UPDATE EMP SET SALARY = (SELECT avg(SALARY) FROM EMP)
WHERE EMP_ID = 'E10001'
```

Notes

- Update is supported for primitive types, non-nested Tuple types, and non-nested user-defined types. Update is also supported for values in non-nested Map types. The driver does not support updates on List types, Set types, or keys in Map types because the values in each are part of the primary key of their respective child tables and primary key columns cannot be updated. If an Update is attempted when not allowed, the driver issues the following error message:

  `[DataDirect][Cassandra ODBC Driver][Cassandra]syntax error or access rule violation: UPDATE not permitted for column: column_name`

  See "Complex type normalization" for details.

- Update is supported for Counter columns when all the other columns in the row comprise that row's primary key. The Counter column itself is the only updatable field in the row. When updating a Counter column on an existing row, the Counter column is updated according to the increment (or decrement) specified in the SQL statement. When updating a Counter column for which there is no existing row, the values of the columns that comprise the row's primary key are inserted into the table alongside the value of the Counter column.

  For example, consider the following table.

  ```
  CREATE TABLE page_view_counts (
    counter_value counter,
    url_name varchar,
    page_name varchar,
    PRIMARY KEY (url_name, page_name));
  ```

  The following Update can be performed on the `page_view_counts` table.

  ```
  UPDATE PAGE_VIEW_COUNTS
     SET COUNTER_VALUE=COUNTER_VALUE + 1
     WHERE URL_NAME = 'www.progress.com' AND page_name = 'home'
  ```

  This Update would provide the following output.

Note: Cassandra initially assigns a value of `0` (zero) to Counter columns. An increment or decrement can be specified in the SQL statement.

```
 URL_NAME          | PAGE_NAME  | COUNTER_VALUE
-------------------+------------+---------------
 www.progress.com  |       home |             1
```

- A Where clause can be used to restrict which rows are updated.
- To enable Insert, Update, and Delete, set the Read Only (ReadOnly) connection option to `0` (Disabled).

See also

- Where clause on page 129
- Subqueries on page 144
- Complex type normalization on page 18
- Native and Refresh escape clauses on page 120

SQL expressions

An expression is a combination of one or more values, operators, and SQL functions that evaluate to a value. You can use expressions in the Where, and Having of Select statements; and in the Set clauses of Update statements.

Expressions enable you to use mathematical operations as well as character string manipulation operators to form complex queries.

The driver supports both unquoted and quoted identifiers. An unquoted identifier must start with an ASCII alpha character and can be followed by zero or more ASCII alphanumeric characters. Unquoted identifiers are converted to uppercase before being used.

Quoted identifiers must be enclosed in double quotation marks (""). A quoted identifier can contain any Unicode character including the space character. The driver recognizes the Unicode escape sequence \uxxxx as a Unicode character. You can specify a double quotation mark in a quoted identifier by escaping it with a double quotation mark.

The maximum length of both quoted and unquoted identifiers is 128 characters.

Valid expression elements are:

- Column names
- Literals
- Operators
- Functions

Column names

The most common expression is a simple column name. You can combine a column name with other expression elements.

Literals

Literals are fixed data values. For example, in the expression `PRICE * 1.05`, the value 1.05 is a constant. Literals are classified into types, including the following:

- Binary
- Character string
- Date
- Floating point
- Integer
- Numeric
- Time
- Timestamp

The following table describes the literal format for supported SQL data types.

Table 18: Literal Syntax Examples

SQL Type	Literal Syntax	Example
BIGINT	*n* where *n* is any valid integer value in the range of the INTEGER data type	`12 or -34 or 0`
BOOLEAN	Min Value: 0 Max Value: 1	`0` `1`
DATE	DATE'*date*'	`'2010-05-21'`
DATETIME	TIMESTAMP'*ts*'	`'2010-05-21 18:33:05.025'`

DECIMAL	*n.f*
	where:
	n
	is the integral part
	f
	is the fractional part
		`0.25`
		`3.1415`
		`-7.48`

DOUBLE	*n.f*E*x* where: *n* is the integral part *f* is the fractional part *x* is the exponent	`1.2E0` or `2.5E40` or `-3.45E2` or `5.67E-4`

SQL Type	Literal Syntax	Example
INTEGER	*n* where *n* is a valid integer value in the range of the INTEGER data type	`12` or `-34` or `0`
LONGVARBINARY	X'*hex_value*'	`'000482ff'`
LONGVARCHAR	'*value*'	`'This is a string literal'`
TIME	TIME'*time*'	`'2010-05-21 18:33:05.025'`
VARCHAR	'*value*'	`'This is a string literal'`

Character string literals

Text specifies a character string literal. A character string literal must be enclosed in single quotation marks. To represent one single quotation mark within a literal, you must enter two single quotation marks. When the data in the fields is returned to the client, trailing blanks are stripped.

A character string literal can have a maximum length of 32 KB, that is, (32*1024) bytes.

Example

```
'Hello'
'Jim''s friend is Joe'
```

Numeric literals

Unquoted numeric values are treated as numeric literals. If the unquoted numeric value contains a decimal point or exponent, it is treated as a real literal; otherwise, it is treated as an integer literal.

Example

```
+1894.1204
```

Binary literals

Binary literals are represented with single quotation marks. The valid characters in a binary literal are 0-9, a-f, and A-F.

Example

```
'00af123d'
```

Date/time literals

Date and time literal values are enclosed in single quotion marks ('*value*').

- The format for a Date literal is DATE'*date*'.

- The format for a Time literal is TIME'*time*'.
- The format for a Timestamp literal is TIMESTAMP'*ts*'.

Integer literals

Integer literals are represented by a string of numbers that are not enclosed in quotation marks and do not contain decimal points.

Notes

- Integer constants must be whole numbers; they cannot contain decimals.
- Integer literals can start with sign characters (+/-).

Example

```
1994 or -2
```

Operators

This section describes the operators that can be used in SQL expressions.

Unary operator

A unary operator operates on only one operand.

operator operand

Binary operator

A binary operator operates on two operands.

operand1 operator operand2

If an operator is given a null operand, the result is always null. The only operator that does not follow this rule is concatenation (||).

Arithmetic operators

You can use an arithmetic operator in an expression to negate, add, subtract, multiply, and divide numeric values. The result of this operation is also a numeric value. The + and - operators are also supported in date/time fields to allow date arithmetic. The following table lists the supported arithmetic operators.

Table 19: Arithmetic Operators

Operator	Purpose	Example
+ -	Denotes a positive or negative expression. These are unary operators.	`SELECT * FROM EMP WHERE COMM = -1`

Operator	Purpose	Example
* /	Multiplies, divides. These are binary operators.	`UPDATE EMP SET SAL = SAL + SAL * 0.10`
+ -	Adds, subtracts. These are binary operators.	`SELECT SAL + COMM FROM EMP WHERE EMPNO > 100`

Concatenation operator

The concatenation operator manipulates character strings. The following table lists the only supported concatenation operator.

Table 20: Concatenation Operator

Operator	Purpose	Example
||	Concatenates character strings.	`SELECT 'Name is' || ENAME FROM EMP`

The result of concatenating two character strings is the data type VARCHAR.

Comparison operators

Comparison operators compare one expression to another. The result of such a comparison can be TRUE, FALSE, or UNKNOWN (if one of the operands is NULL). The driver considers the UNKNOWN result as FALSE.

The following table lists the supported comparison operators.

Table 21: Comparison Operators

Operator	Purpose	Example
=	Equality test.	`SELECT * FROM EMP WHERE SAL = 1500`
!=<>	Inequality test.	`SELECT * FROM EMP WHERE SAL != 1500`
><	“Greater than" and "less than" tests.	`SELECT * FROM EMP WHERE SAL > 1500 SELECT * FROM EMP WHERE SAL < 1500`
>=<=	“Greater than or equal to" and "less than or equal to" tests.	`SELECT * FROM EMP WHERE SAL >= 1500 SELECT * FROM EMP WHERE SAL <= 1500`
ESCAPE clause in LIKE operator LIKE ’pattern string’ ESCAPE ’c’	The Escape clause is supported in the LIKE predicate to indicate the escape character. Escape characters are used in the pattern string to indicate that any wildcard character that is after the escape	`SELECT * FROM EMP WHERE ENAME LIKE 'J%\_%' ESCAPE '\'` This matches all records with names that start with letter 'J' and have the '_' character in them.

Operator	Purpose	Example
	character in the pattern string should be treated as a regular character. The default escape character is backslash (\).	`SELECT * FROM EMP WHERE ENAME LIKE 'JOE\_JOHN' ESCAPE '\'` This matches only records with name 'JOE_JOHN'.
[NOT] IN	“Equal to any member of" test.	`SELECT * FROM EMP WHERE JOB IN ('CLERK','ANALYST') SELECT * FROM EMP WHERE SAL IN (SELECT SAL FROM EMP WHERE DEPTNO = 30)`
[NOT] BETWEEN x AND y	"Greater than or equal to x" and "less than or equal to y."	`SELECT * FROM EMP WHERE SAL BETWEEN 2000 AND 3000`
EXISTS	Tests for existence of rows in a subquery.	`SELECT EMPNO, ENAME, DEPTNO FROM EMP E WHERE EXISTS (SELECT DEPTNO FROM DEPT WHERE E.DEPTNO = DEPT.DEPTNO)`
IS [NOT] NULL	Tests whether the value of the column or expression is NULL.	`SELECT * FROM EMP WHERE ENAME IS NOT NULL SELECT * FROM EMP WHERE ENAME IS NULL`

Logical operators

A logical operator combines the results of two component conditions to produce a single result or to invert the result of a single condition. The following table lists the supported logical operators.

Table 22: Logical Operators

Operator	Purpose	Example
NOT	Returns TRUE if the following condition is FALSE. Returns FALSE if it is TRUE. If it is UNKNOWN, it remains UNKNOWN.	`SELECT * FROM EMP WHERE NOT (JOB IS NULL)` `SELECT * FROM emp WHERE NOT (SAL BETWEEN 1000 AND 2000)`
AND	Returns TRUE if both component conditions are TRUE. Returns FALSE if either is FALSE; otherwise, returns UNKNOWN.	`SELECT * FROM EMP WHERE JOB = 'CLERK' AND DEPTNO = 10`
OR	Returns TRUE if either component condition is TRUE. Returns FALSE if both are FALSE; otherwise, returns UNKNOWN.	`SELECT * FROM EMP WHERE JOB = 'CLERK' OR DEPTNO = 10`

Example

In the Where clause of the following Select statement, the AND logical operator is used to ensure that managers earning more than $1000 a month are returned in the result:

```
SELECT * FROM EMP WHERE JOBTITLE = MANAGER AND SAL > 1000
```

Operator precedence

As expressions become more complex, the order in which the expressions are evaluated becomes important. The following table shows the order in which the operators are evaluated. The operators in the first line are evaluated first, then those in the second line, and so on. Operators in the same line are evaluated left to right in the expression. You can change the order of precedence by using parentheses. Enclosing expressions in parentheses forces them to be evaluated together.

Table 23: Operator Precedence

Precedence	Operator
1	+ (Positive), - (Negative)
2	*(Multiply), / (Division)
3	+ (Add), - (Subtract)
4	|| (Concatenate)
5	=, >, <, >=, <=, <>, != (Comparison operators)
6	NOT, IN, LIKE
7	AND
8	OR

Example A

The query in the following example returns employee records for which the department number is 1 or 2 and the salary is greater than $1000:

```
SELECT * FROM EMP WHERE (DEPTNO = 1 OR DEPTNO = 2) AND SAL > 1000
```

Because parenthetical expressions are forced to be evaluated first, the OR operation takes precedence over AND.

Example B

In the following example, the query returns records for all the employees in department 1, but only employees whose salary is greater than $1000 in department 2.

```
SELECT * FROM EMP WHERE DEPTNO = 1 OR DEPTNO = 2 AND SAL > 1000
```

The AND operator takes precedence over OR, so that the search condition in the example is equivalent to the expression `DEPTNO = 1 OR (DEPTNO = 2 AND SAL > 1000)`.

Functions

The driver supports a number of functions that you can use in expressions.

Refer to "Scalar functions" in the *Progress DataDirect for ODBC Drivers Reference* for more information.

Conditions

A condition specifies a combination of one or more expressions and logical operators that evaluates to either TRUE, FALSE, or UNKNOWN. You can use a condition in the Where clause of the Delete, Select, and Update statements; and in the Having clauses of Select statements. The following describes supported conditions.

Table 24: Conditions

Condition	Description
Simple comparison	Specifies a comparison with expressions or subquery results. `= , !=, <>, < , >, >=, <=`
Group comparison	Specifies a comparison with any or all members in a list or subquery. `[= , !=, <>, < , >, >=, <=] [ANY, ALL, SOME]`
Membership	Tests for membership in a list or subquery. `[NOT] IN`
Range	Tests for inclusion in a range. `[NOT] BETWEEN`

Condition	Description
NULL	Tests for nulls. `IS NULL, IS NOT NULL`
EXISTS	Tests for existence of rows in a subquery. `[NOT] EXISTS`
LIKE	Specifies a test involving pattern matching. `[NOT] LIKE`
Compound	Specifies a combination of other conditions. `CONDITION [AND/OR] CONDITION`

Subqueries

A query is an operation that retrieves data from one or more tables or views. In this reference, a top-level query is called a Select statement, and a query nested within a Select statement is called a subquery.

A subquery is a query expression that appears in the body of another expression such as a Select, an Update, or a Delete statement. In the following example, the second Select statement is a subquery:

```
SELECT * FROM EMP WHERE DEPTNO IN (SELECT DEPTNO FROM DEPT)
```

IN predicate

Purpose

The In predicate specifies a set of values against which to compare a result set. If the values are being compared against a subquery, only a single column result set is returned.

Syntax

value [NOT] IN (*value1, value2,...*)

OR

value [NOT] IN (*subquery*)

Example

```
SELECT * FROM EMP WHERE DEPTNO IN
(SELECT DEPTNO FROM DEPT WHERE DNAME <> 'Sales')
```

EXISTS predicate

Purpose

The Exists predicate is true only if the cardinality of the subquery is greater than 0; otherwise, it is false.

Syntax

```
EXISTS (subquery)
```

Example

```
SELECT EMPNO, ENAME, DEPTNO FROM EMP E WHERE EXISTS
(SELECT DEPTNO FROM DEPT WHERE E.DEPTNO = DEPT.DEPTNO)
```

UNIQUE predicate

Purpose

The Unique predicate is used to determine whether duplicate rows exist in a virtual table (one returned from a subquery).

Syntax

```
UNIQUE (subquery)
```

Example

```
SELECT * FROM DEPT D WHERE UNIQUE
(SELECT DEPTNO FROM EMP E WHERE E.DEPTNO = D.DEPTNO)
```

Correlated subqueries

Purpose

A correlated subquery is a subquery that references a column from a table referred to in the parent statement. A correlated subquery is evaluated once for each row processed by the parent statement. The parent statement can be a Select, Update, or Delete statement.

A correlated subquery answers a multiple-part question in which the answer depends on the value in each row processed by the parent statement. For example, you can use a correlated subquery to determine which employees earn more than the average salaries for their departments. In this case, the correlated subquery specifically computes the average salary for each department.

Syntax

```
SELECT select_list
   FROM table1 t_alias1
   WHERE expr rel_operator
   (SELECT column_list
       FROM table2 t_alias2
```

```
       WHERE t_alias1.columnrel_operatort_alias2.column)
UPDATE table1 t_alias1
   SET column =
   (SELECT expr
       FROM table2 t_alias2
        WHERE t_alias1.column = t_alias2.column)
DELETE FROM table1 t_alias1
   WHERE column rel_operator
   (SELECT expr
       FROM table2 t_alias2
       WHERE t_alias1.column = t_alias2.column)
```

Notes

- Correlated column names in correlated subqueries must be explicitly qualified with the table name of the parent.

Example A

The following statement returns data about employees whose salaries exceed their department average. This statement assigns an alias to emp, the table containing the salary information, and then uses the alias in a correlated subquery:

```
SELECT deptno, ename, sal FROM emp x WHERE sal >
  (SELECT AVG(sal) FROM emp WHERE x.deptno = deptno)
  ORDER BY deptno
```

Example B

This is an example of a correlated subquery that returns row values:

```
SELECT * FROM dept "outer" WHERE 'manager' IN
  (SELECT managername FROM emp
  WHERE "outer".deptno = emp.deptno)
```

Example C

This is an example of finding the department number (deptno) with multiple employees:

```
SELECT * FROM dept main WHERE 1 <
  (SELECT COUNT(*) FROM emp WHERE deptno = main.deptno)
```

Example D

This is an example of correlating a table with itself:

```
SELECT deptno, ename, sal FROM emp x WHERE sal >
  (SELECT AVG(sal) FROM emp WHERE x.deptno = deptno)
```